UM1850
User manual

Description of STM32F1 HAL and Low-layer drivers

Introduction

STMCube™ is an STMicroelectronics original initiative to make developers' lives easier by reducing development efforts, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube Version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF1 for STM32F1 series)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio
  - Low Layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities delivered with a full set of examples.

The HAL driver layer provides a generic multi-instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks).

The HAL driver 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 line or part number. 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, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors.

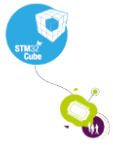
The HAL drivers are feature-oriented instead of IP-oriented. As an example, the timer APIs are split into several categories following the IP functions: basic timer, capture, pulse width modulation (PWM), etc.. The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking contributes to enhance the firmware robustness. Run-time detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide atomic operations that must be called following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter or data pointers: all operations are performed by changing the associated peripheral registers content. Contrary to the HAL, the LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or complex upper level stack (such as FSMC, USB, SDMMC).

The HAL and LL are complementary and cover a wide range of applications requirements:

- The HAL offers high-level and feature-oriented APIs, with a high-portability level. They hide the MCU and peripheral complexity to end-user.
- The LL offers low-level APIs at registers level, with better optimization but less portability. They require deep knowledge of the MCU and peripherals specifications.

The source code of HAL and LL drivers 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.



Contents

1 Acronyms and definitions 24
2 Overview of HAL drivers 26
2.1 HAL and user-application files 26
2.1.1 HAL driver files 26
2.1.2 User-application files 27
2.2 HAL data structures 29
2.2.1 Peripheral handle structures 29
2.2.2 Initialization and configuration structure 30
2.2.3 Specific process structures 31
2.3 API classification 31
2.4 Devices supported by HAL drivers 32
2.5 HAL driver rules 38
2.5.1 HAL API naming rules 38
2.5.2 HAL general naming rules 39
2.5.3 HAL interrupt handler and callback functions 40
2.6 HAL generic APIs 41
2.7 HAL extension APIs 42
2.7.1 HAL extension model overview 42
2.7.2 HAL extension model cases 42
2.8 File inclusion model 45
2.9 HAL common resources 46
2.10 HAL configuration 46
2.11 HAL system peripheral handling 47
2.11.1 Clock 47
2.11.2 GPIOs 48
2.11.3 Cortex NVIC and SysTick timer 50
2.11.4 PWR 50
2.11.5 EXTI 50
2.11.6 DMA 52
2.12 How to use HAL drivers 53
2.12.1 HAL usage models 53
2.12.2 HAL initialization 54
2.12.3 HAL IO operation process 56
2.12.4 Timeout and error management 59

3 Overview of Low Layer drivers 63
3.1 Low Layer files 63
3.2 Overview of Low Layer APIs and naming rules 65
3.2.1 Peripheral initialization functions 65
3.2.2 Peripheral register-level configuration functions 69
4 Cohabiting of HAL and LL 71
4.1 Low Layer driver used in standalone mode 71
4.2 Mixed use of Low Layer APIs and HAL drivers 71
5 HAL System Driver 72
5.1 HAL Firmware driver API description 72
5.1.1 How to use this driver 72
5.1.2 Initialization and de-initialization functions 72
5.1.3 HAL Control functions 72
5.1.4 Detailed description of functions 73
5.2 HAL Firmware driver defines 77
5.2.1 HAL 77
6 HAL ADC Generic Driver 79
6.1 ADC Firmware driver registers structures 79
6.1.1 ADC_InitTypeDef 79
6.1.2 ADC_ChannelConfTypeDef 80
6.1.3 ADC_AnalogWDGConfTypeDef 80
6.1.4 ADC_HandleTypeDef 81
6.2 ADC Firmware driver API description 81
6.2.1 ADC peripheral features 81
6.2.2 How to use this driver 82
6.2.3 Initialization and de-initialization functions 84
6.2.4 IO operation functions 85
6.2.5 Peripheral Control functions 85
6.2.6 Peripheral State and Errors functions 85
6.2.7 Detailed description of functions 86
6.3 ADC Firmware driver defines 92
6.3.1 ADC 92
7 HAL ADC Extension Driver 100
7.1 ADCEx Firmware driver registers structures 100
7.1.1 ADC_InjectionConfTypeDef 100
7.1.2 ADC_MultiModeTypeDef 101

7.2 ADCEx Firmware driver API description 102
7.2.1 IO operation functions 102
7.2.2 Peripheral Control functions 102
7.2.3 Detailed description of functions 103
7.3 ADCEx Firmware driver defines 106
7.3.1 ADCEx 106
8 HAL CAN Generic Driver 110
8.1 CAN Firmware driver registers structures 110
8.1.1 CAN_InitTypeDef 110
8.1.2 CanTxMsgTypeDef 111
8.1.3 CanRxMsgTypeDef 111
8.1.4 CAN_HandleTypeDef 112
8.2 CAN Firmware driver API description 112
8.2.1 How to use this driver 112
8.2.2 Initialization and de-initialization functions 113
8.2.3 IO operation functions 114
8.2.4 Peripheral State and Error functions 114
8.2.5 Detailed description of functions 114
8.3 CAN Firmware driver defines 118
8.3.1 CAN 118
9 HAL CAN Extension Driver 127
9.1 CANEx Firmware driver registers structures 127
9.1.1 CAN_FilterConfTypeDef 127
10 HAL CEC Generic Driver 128
10.1 CEC Firmware driver registers structures 128
10.1.1 CEC_InitTypeDef 128
10.1.2 CEC_HandleTypeDef 128
10.2 CEC Firmware driver API description 129
10.2.1 How to use this driver 129
10.2.2 Initialization and Configuration functions 129
10.2.3 IO operation functions 130
10.2.4 Peripheral Control function 130
10.2.5 Detailed description of functions 131
10.3 CEC Firmware driver defines 133
10.3.1 CEC 133
11 HAL CORTEX Generic Driver 139

11.1 CORTEX Firmware driver registers structures 139
11.1.1 MPU_Region_InitTypeDef 139
11.2 CORTEX Firmware driver API description 140
11.2.1 How to use this driver 140
11.2.2 Initialization and de-initialization functions 140
11.2.3 Peripheral Control functions 141
11.2.4 Detailed description of functions 141
11.3 CORTEX Firmware driver defines 146
11.3.1 CORTEX 146
12 HAL CRC Generic Driver 149
12.1 CRC Firmware driver registers structures 149
12.1.1 CRC_HandleTypeDef 149
12.2 CRC Firmware driver API description 149
12.2.1 How to use this driver 149
12.2.2 Initialization and de-initialization functions 149
12.2.3 Peripheral Control functions 149
12.2.4 Peripheral State functions 150
12.2.5 Detailed description of functions 150
12.3 CRC Firmware driver defines 151
12.3.1 CRC 151
13 HAL DAC Generic Driver 153
13.1 DAC Firmware driver registers structures 153
13.1.1 DAC_HandleTypeDef 153
13.1.2 DAC_ChannelConfTypeDef 153
13.2 DAC Firmware driver API description 153
13.2.1 DAC Peripheral features 153
13.2.2 How to use this driver 155
13.2.3 Initialization and de-initialization functions 156
13.2.4 IO operation functions 156
13.2.5 Peripheral Control functions 157
13.2.6 Peripheral State and Errors functions 157
13.2.7 Detailed description of functions 157
13.3 DAC Firmware driver defines 162
13.3.1 DAC 162
14 HAL DAC Extension Driver 164
14.1 DACEx Firmware driver API description 164

14.1.1 How to use this driver 164
14.1.2 Extended features functions 164
14.1.3 Detailed description of functions 164
14.2 DACEx Firmware driver defines 168
14.2.1 DACEx 168
15 HAL DMA Generic Driver 170
15.1 DMA Firmware driver registers structures 170
15.1.1 DMA_InitTypeDef 170
15.1.2 __DMA_HandleTypeDef 170
15.2 DMA Firmware driver API description 171
15.2.1 How to use this driver 171
15.2.2 Initialization and de-initialization functions 172
15.2.3 IO operation functions 172
15.2.4 Peripheral State and Errors functions 173
15.2.5 Detailed description of functions 173
15.3 DMA Firmware driver defines 176
15.3.1 DMA 176
16 HAL DMA Extension Driver 181
16.1 DMAEx Firmware driver defines 181
16.1.1 DMAEx 181
17 HAL ETH Generic Driver 183
17.1 ETH Firmware driver registers structures 183
17.1.1 ETH_InitTypeDef 183
17.1.2 ETH_MACInitTypeDef 183
17.1.3 ETH_DMAInitTypeDef 186
17.1.4 ETH_DMADescTypeDef 187
17.1.5 ETH_DMARxFrameInfos 187
17.1.6 ETH_HandleTypeDef 188
17.2 ETH Firmware driver API description 188
17.2.1 How to use this driver 188
17.2.2 Initialization and de-initialization functions 189
17.2.3 IO operation functions 189
17.2.4 Peripheral Control functions 190
17.2.5 Peripheral State functions 190
17.2.6 Detailed description of functions 190
17.3 ETH Firmware driver defines 195
17.3.1 ETH 195

18 HAL FLASH Generic Driver 221

18.1 FLASH Firmware driver registers structures 221

18.1.1 FLASH_ProcessTypeDef 221

18.2 FLASH Firmware driver API description 221

18.2.1 FLASH peripheral features 221

18.2.2 How to use this driver 221

18.2.3 Peripheral Control functions 222

18.2.4 Peripheral Errors functions 222

18.2.5 Detailed description of functions 222

18.3 FLASH Firmware driver defines 225

18.3.1 FLASH 225

19 HAL FLASH Extension Driver 230

19.1 FLASHEx Firmware driver registers structures 230

19.1.1 FLASH_EraseInitTypeDef 230

19.1.2 FLASH_OBProgramInitTypeDef 230

19.2 FLASHEx Firmware driver API description 231

19.2.1 FLASH Erasing Programming functions 231

19.2.2 Option Bytes Programming functions 231

19.2.3 Detailed description of functions 231

19.3 FLASHEx Firmware driver defines 233

19.3.1 FLASHEx 233

20 HAL GPIO Generic Driver 236

20.1 GPIO Firmware driver registers structures 236

20.1.1 GPIO_InitTypeDef 236

20.2 GPIO Firmware driver API description 236

20.2.1 GPIO Peripheral features 236

20.2.2 How to use this driver 237

20.2.3 Initialization and de-initialization functions 237

20.2.4 IO operation functions 237

20.2.5 Detailed description of functions 238

20.3 GPIO Firmware driver defines 240

20.3.1 GPIO 240

21 HAL GPIO Extension Driver 243

21.1 GPIOEx Firmware driver API description 243

21.1.1 GPIO Peripheral extension features 243

21.1.2 How to use this driver 243

21.1.3 Extended features functions ... 243
21.1.4 Detailed description of functions ... 243
21.2 GPIOEx Firmware driver defines ... 244
21.2.1 GPIOEx ... 244
22 HAL HCD Generic Driver ... 259
22.1 HCD Firmware driver registers structures ... 259
22.1.1 HCD_HandleTypeDef ... 259
22.2 HCD Firmware driver API description ... 259
22.2.1 How to use this driver ... 259
22.2.2 Initialization and de-initialization functions ... 259
22.2.3 IO operation functions ... 260
22.2.4 Peripheral Control functions ... 260
22.2.5 Peripheral State functions ... 260
22.2.6 Detailed description of functions ... 260
22.3 HCD Firmware driver defines ... 265
22.3.1 HCD ... 265
23 HAL I2C Generic Driver ... 266
23.1 I2C Firmware driver registers structures ... 266
23.1.1 I2C_InitTypeDef ... 266
23.1.2 I2C_HandleTypeDef ... 266
23.2 I2C Firmware driver API description ... 267
23.2.1 How to use this driver ... 267
23.2.2 Initialization and de-initialization functions ... 272
23.2.3 IO operation functions ... 272
23.2.4 Peripheral State, Mode and Error functions ... 274
23.2.5 Detailed description of functions ... 274
23.3 I2C Firmware driver defines ... 286
23.3.1 I2C ... 286
24 HAL I2S Generic Driver ... 292
24.1 I2S Firmware driver registers structures ... 292
24.1.1 I2S_InitTypeDef ... 292
24.1.2 __I2S_HandleTypeDef ... 292
24.2 I2S Firmware driver API description ... 293
24.2.1 How to use this driver ... 293
24.2.2 Initialization and de-initialization functions ... 295
24.2.3 IO operation functions ... 295
24.2.4 Peripheral State and Errors functions ... 296

24.2.5 Detailed description of functions ... 296
24.3 I2S Firmware driver defines ... 302
24.3.1 I2S ... 302
25 HAL IRDA Generic Driver ... 307
25.1 IRDA Firmware driver registers structures ... 307
25.1.1 IRDA_InitTypeDef ... 307
25.1.2 IRDA_HandleTypeDef ... 307
25.2 IRDA Firmware driver API description ... 308
25.2.1 How to use this driver ... 308
25.2.2 Initialization and Configuration functions ... 310
25.2.3 IO operation functions ... 311
25.2.4 Peripheral State and Errors functions ... 312
25.2.5 Detailed description of functions ... 312
25.3 IRDA Firmware driver defines ... 320
25.3.1 IRDA ... 320
26 HAL IWDG Generic Driver ... 326
26.1 IWDG Firmware driver registers structures ... 326
26.1.1 IWDG_InitTypeDef ... 326
26.1.2 IWDG_HandleTypeDef ... 326
26.2 IWDG Firmware driver API description ... 326
26.2.1 IWDG Generic features ... 326
26.2.2 How to use this driver ... 327
26.2.3 Initialization and Start functions ... 327
26.2.4 IO operation functions ... 327
26.2.5 Detailed description of functions ... 327
26.3 IWDG Firmware driver defines ... 328
26.3.1 IWDG ... 328
27 HAL NAND Generic Driver ... 330
27.1 NAND Firmware driver registers structures ... 330
27.1.1 NAND_IDTypeDef ... 330
27.1.2 NAND_AddressTypeDef ... 330
27.1.3 NAND_DeviceConfigTypeDef ... 330
27.1.4 NAND_HandleTypeDef ... 331
27.2 NAND Firmware driver API description ... 331
27.2.1 How to use this driver ... 331
27.2.2 NAND Initialization and de-initialization functions ... 332

27.2.3 NAND Input and Output functions 332
27.2.4 NAND Control functions 333
27.2.5 NAND State functions 333
27.2.6 Detailed description of functions 333
27.3 NAND Firmware driver defines 339
27.3.1 NAND 339
28 HAL NOR Generic Driver 340
28.1 NOR Firmware driver registers structures 340
28.1.1 NOR_IDTypeDef 340
28.1.2 NOR_CFITypeDef 340
28.1.3 NOR_HandleTypeDef 340
28.2 NOR Firmware driver API description 341
28.2.1 How to use this driver 341
28.2.2 NOR Initialization and de_initialization functions 341
28.2.3 NOR Input and Output functions 342
28.2.4 NOR Control functions 342
28.2.5 NOR State functions 342
28.2.6 Detailed description of functions 342
28.3 NOR Firmware driver defines 346
28.3.1 NOR 346
29 HAL PCCARD Generic Driver 348
29.1 PCCARD Firmware driver registers structures 348
29.1.1 PCCARD_HandleTypeDef 348
29.2 PCCARD Firmware driver API description 348
29.2.1 How to use this driver 348
29.2.2 PCCARD Initialization and de-initialization functions 349
29.2.3 PCCARD Input Output and memory functions 349
29.2.4 PCCARD Peripheral State functions 349
29.2.5 Detailed description of functions 349
29.3 PCCARD Firmware driver defines 353
29.3.1 PCCARD 353
30 HAL PCD Generic Driver 354
30.1 PCD Firmware driver registers structures 354
30.1.1 PCD_HandleTypeDef 354
30.2 PCD Firmware driver API description 354
30.2.1 How to use this driver 354
30.2.2 Initialization and de-initialization functions 355

30.2.3 IO operation functions ... 355
30.2.4 Peripheral Control functions ... 355
30.2.5 Peripheral State functions ... 356
30.2.6 Detailed description of functions ... 356
30.3 PCD Firmware driver defines ... 362
30.3.1 PCD ... 362
31 HAL PCD Extension Driver ... 364
31.1 PCDEx Firmware driver API description ... 364
31.1.1 Extended Peripheral Control functions ... 364
31.1.2 Detailed description of functions ... 364
32 HAL PWR Generic Driver ... 365
32.1 PWR Firmware driver registers structures ... 365
32.1.1 PWR_PVDTypeDef ... 365
32.2 PWR Firmware driver API description ... 365
32.2.1 Initialization and de-initialization functions ... 365
32.2.2 Peripheral Control functions ... 365
32.2.3 Detailed description of functions ... 367
32.3 PWR Firmware driver defines ... 371
32.3.1 PWR ... 371
33 HAL RCC Generic Driver ... 376
33.1 RCC Firmware driver registers structures ... 376
33.1.1 RCC_PLLInitTypeDef ... 376
33.1.2 RCC_ClkInitTypeDef ... 376
33.2 RCC Firmware driver API description ... 377
33.2.1 RCC specific features ... 377
33.2.2 RCC Limitations ... 377
33.2.3 Initialization and de-initialization functions ... 377
33.2.4 Peripheral Control functions ... 378
33.2.5 Detailed description of functions ... 378
33.3 RCC Firmware driver defines ... 383
33.3.1 RCC ... 383
34 HAL RCC Extension Driver ... 398
34.1 RCCEx Firmware driver registers structures ... 398
34.1.1 RCC_OscInitTypeDef ... 398
34.1.2 RCC_PeriphCLKInitTypeDef ... 398
34.2 RCCEx Firmware driver API description ... 399

34.2.1 Extended Peripheral Control functions ... 399
34.2.2 Detailed description of functions ... 399
34.3 RCCEx Firmware driver defines ... 400
34.3.1 RCCEx ... 400
35 HAL RTC Generic Driver ... 409
35.1 RTC Firmware driver registers structures ... 409
35.1.1 RTC_TimeTypeDef ... 409
35.1.2 RTC_AlarmTypeDef ... 409
35.1.3 RTC_InitTypeDef ... 409
35.1.4 RTC_DateTypeDef ... 410
35.1.5 RTC_HandleTypeDef ... 410
35.2 RTC Firmware driver API description ... 410
35.2.1 How to use this driver ... 410
35.2.2 WARNING: Drivers Restrictions ... 411
35.2.3 Backup Domain Operating Condition ... 411
35.2.4 Backup Domain Reset ... 412
35.2.5 Backup Domain Access ... 412
35.2.6 RTC and low power modes ... 412
35.2.7 Initialization and de-initialization functions ... 412
35.2.8 RTC Time and Date functions ... 413
35.2.9 RTC Alarm functions ... 413
35.2.10 Peripheral State functions ... 413
35.2.11 Peripheral Control functions ... 413
35.2.12 Detailed description of functions ... 413
35.3 RTC Firmware driver defines ... 418
35.3.1 RTC ... 418
36 HAL RTC Extension Driver ... 425
36.1 RTCEx Firmware driver registers structures ... 425
36.1.1 RTC_TamperTypeDef ... 425
36.2 RTCEx Firmware driver API description ... 425
36.2.1 RTC Tamper functions ... 425
36.2.2 RTC Second functions ... 425
36.2.3 Extension Peripheral Control functions ... 425
36.2.4 Detailed description of functions ... 426
36.3 RTCEx Firmware driver defines ... 429
36.3.1 RTCEx ... 429
37 HAL SD Generic Driver ... 436

37.1 SD Firmware driver registers structures 436
37.1.1 HAL_SD_CardInfoTypeDef 436
37.1.2 SD_HandleTypeDef 436
37.1.3 HAL_SD_CardCSDTypeDef 437
37.1.4 HAL_SD_CardCIDTypeDef 439
37.1.5 HAL_SD_CardStatusTypeDef 440
37.2 SD Firmware driver API description 441
37.2.1 How to use this driver 441
37.2.2 Initialization and de-initialization functions 444
37.2.3 IO operation functions 444
37.2.4 Peripheral Control functions 444
37.2.5 Detailed description of functions 445
37.3 SD Firmware driver defines 451
37.3.1 SD 451
38 HAL SMARTCARD Generic Driver 459
38.1 SMARTCARD Firmware driver registers structures 459
38.1.1 SMARTCARD_InitTypeDef 459
38.1.2 SMARTCARD_HandleTypeDef 460
38.2 SMARTCARD Firmware driver API description 461
38.2.1 How to use this driver 461
38.2.2 Initialization and Configuration functions 463
38.2.3 IO operation functions 464
38.2.4 Peripheral State and Errors functions 465
38.2.5 Detailed description of functions 465
38.3 SMARTCARD Firmware driver defines 472
38.3.1 SMARTCARD 472
39 HAL SPI Generic Driver 480
39.1 SPI Firmware driver registers structures 480
39.1.1 SPI_InitTypeDef 480
39.1.2 __SPI_HandleTypeDef 481
39.2 SPI Firmware driver API description 482
39.2.1 How to use this driver 482
39.2.2 Initialization and de-initialization functions 482
39.2.3 IO operation functions 483
39.2.4 Peripheral State and Errors functions 484
39.2.5 Detailed description of functions 484
39.3 SPI Firmware driver defines 491

39.3.1 SPI ... 491
40 HAL SRAM Generic Driver ... 496
40.1 SRAM Firmware driver registers structures ... 496
40.1.1 SRAM_HandleTypeDef ... 496
40.2 SRAM Firmware driver API description ... 496
40.2.1 How to use this driver ... 496
40.2.2 SRAM Initialization and de_initialization functions ... 497
40.2.3 SRAM Input and Output functions ... 497
40.2.4 SRAM Control functions ... 497
40.2.5 SRAM State functions ... 498
40.2.6 Detailed description of functions ... 498
40.3 SRAM Firmware driver defines ... 502
40.3.1 SRAM ... 502
41 HAL TIM Generic Driver ... 503
41.1 TIM Firmware driver registers structures ... 503
41.1.1 TIM_Base_InitTypeDef ... 503
41.1.2 TIM_OC_InitTypeDef ... 503
41.1.3 TIM_OnePulse_InitTypeDef ... 504
41.1.4 TIM_IC_InitTypeDef ... 505
41.1.5 TIM_Encoder_InitTypeDef ... 505
41.1.6 TIM_ClockConfigTypeDef ... 506
41.1.7 TIM_ClearInputConfigTypeDef ... 506
41.1.8 TIM_SlaveConfigTypeDef ... 507
41.1.9 TIM_HandleTypeDef ... 507
41.2 TIM Firmware driver API description ... 508
41.2.1 TIMER Generic features ... 508
41.2.2 How to use this driver ... 508
41.2.3 Time Base functions ... 509
41.2.4 Time Output Compare functions ... 509
41.2.5 Time PWM functions ... 510
41.2.6 Time Input Capture functions ... 510
41.2.7 Time One Pulse functions ... 511
41.2.8 Time Encoder functions ... 511
41.2.9 IRQ handler management ... 512
41.2.10 Peripheral Control functions ... 512
41.2.11 TIM Callbacks functions ... 512
41.2.12 Peripheral State functions ... 512
41.2.13 Detailed description of functions ... 513

41.3 TIM Firmware driver defines 537
41.3.1 TIM 537
42 HAL TIM Extension Driver 556
42.1 TIMEx Firmware driver registers structures 556
42.1.1 TIM_HallSensor_InitTypeDef 556
42.1.2 TIM_BreakDeadTimeConfigTypeDef 556
42.1.3 TIM_MasterConfigTypeDef 557
42.2 TIMEx Firmware driver API description 557
42.2.1 TIMER Extended features 557
42.2.2 How to use this driver 557
42.2.3 Timer Hall Sensor functions 558
42.2.4 Timer Complementary Output Compare functions 558
42.2.5 Timer Complementary PWM functions 559
42.2.6 Timer Complementary One Pulse functions 559
42.2.7 Peripheral Control functions 559
42.2.8 Extension Callbacks functions 560
42.2.9 Extension Peripheral State functions 560
42.2.10 Detailed description of functions 560
42.3 TIMEx Firmware driver defines 570
42.3.1 TIMEx 570
43 HAL UART Generic Driver 571
43.1 UART Firmware driver registers structures 571
43.1.1 UART_InitTypeDef 571
43.1.2 UART_HandleTypeDef 571
43.2 UART Firmware driver API description 572
43.2.1 How to use this driver 572
43.2.2 Initialization and Configuration functions 574
43.2.3 IO operation functions 575
43.2.4 Peripheral Control functions 576
43.2.5 Peripheral State and Errors functions 577
43.2.6 Detailed description of functions 577
43.3 UART Firmware driver defines 587
43.3.1 UART 587
44 HAL USART Generic Driver 596
44.1 USART Firmware driver registers structures 596
44.1.1 USART_InitTypeDef 596

44.1.2 USART_HandleTypeDef ... 596
44.2 USART Firmware driver API description ... 597
44.2.1 How to use this driver ... 597
44.2.2 Initialization and Configuration functions ... 599
44.2.3 IO operation functions ... 600
44.2.4 Peripheral State and Errors functions ... 601
44.2.5 Detailed description of functions ... 601
44.3 USART Firmware driver defines ... 609
44.3.1 USART ... 609
45 HAL WWDG Generic Driver ... 615
45.1 WWDG Firmware driver registers structures ... 615
45.1.1 WWDG_InitTypeDef ... 615
45.1.2 WWDG_HandleTypeDef ... 615
45.2 WWDG Firmware driver API description ... 615
45.2.1 WWDG specific features ... 615
45.2.2 How to use this driver ... 616
45.2.3 Initialization and Configuration functions ... 616
45.2.4 IO operation functions ... 617
45.2.5 Detailed description of functions ... 617
45.3 WWDG Firmware driver defines ... 618
45.3.1 WWDG ... 618
46 LL ADC Generic Driver ... 621
46.1 ADC Firmware driver registers structures ... 621
46.1.1 LL_ADC_CommonInitTypeDef ... 621
46.1.2 LL_ADC_InitTypeDef ... 621
46.1.3 LL_ADC_REG_InitTypeDef ... 621
46.1.4 LL_ADC_INJ_InitTypeDef ... 622
46.2 ADC Firmware driver API description ... 623
46.2.1 Detailed description of functions ... 623
46.3 ADC Firmware driver defines ... 669
46.3.1 ADC ... 669
47 LL BUS Generic Driver ... 697
47.1 BUS Firmware driver API description ... 697
47.1.1 Detailed description of functions ... 697
47.2 BUS Firmware driver defines ... 709
47.2.1 BUS ... 709

48 LL CORTEX Generic Driver 712
48.1 CORTEX Firmware driver API description 712
48.1.1 Detailed description of functions 712
48.2 CORTEX Firmware driver defines 719
48.2.1 CORTEX 719
49 LL CRC Generic Driver 722
49.1 CRC Firmware driver API description 722
49.1.1 Detailed description of functions 722
49.2 CRC Firmware driver defines 723
49.2.1 CRC 723
50 LL DAC Generic Driver 725
50.1 DAC Firmware driver registers structures 725
50.1.1 LL_DAC_InitTypeDef 725
50.2 DAC Firmware driver API description 725
50.2.1 Detailed description of functions 725
50.3 DAC Firmware driver defines 739
50.3.1 DAC 739
51 LL DMA Generic Driver 745
51.1 DMA Firmware driver registers structures 745
51.1.1 LL_DMA_InitTypeDef 745
51.2 DMA Firmware driver API description 746
51.2.1 Detailed description of functions 746
51.3 DMA Firmware driver defines 779
51.3.1 DMA 779
52 LL EXTI Generic Driver 783
52.1 EXTI Firmware driver registers structures 783
52.1.1 LL_EXTI_InitTypeDef 783
52.2 EXTI Firmware driver API description 783
52.2.1 Detailed description of functions 783
52.3 EXTI Firmware driver defines 796
52.3.1 EXTI 796
53 LL GPIO Generic Driver 798
53.1 GPIO Firmware driver registers structures 798
53.1.1 LL_GPIO_InitTypeDef 798
53.2 GPIO Firmware driver API description 798

53.2.1 Detailed description of functions 798
53.3 GPIO Firmware driver defines 831
53.3.1 GPIO 831
54 LL I2C Generic Driver 835
54.1 I2C Firmware driver registers structures 835
54.1.1 LL_I2C_InitTypeDef 835
54.2 I2C Firmware driver API description 835
54.2.1 Detailed description of functions 835
54.3 I2C Firmware driver defines 866
54.3.1 I2C 866
55 LL I2S Generic Driver 871
55.1 I2S Firmware driver registers structures 871
55.1.1 LL_I2S_InitTypeDef 871
55.2 I2S Firmware driver API description 871
55.2.1 Detailed description of functions 871
55.3 I2S Firmware driver defines 885
55.3.1 I2S 885
56 LL IWDG Generic Driver 887
56.1 IWDG Firmware driver API description 887
56.1.1 Detailed description of functions 887
56.2 IWDG Firmware driver defines 890
56.2.1 IWDG 890
57 LL PWR Generic Driver 892
57.1 PWR Firmware driver API description 892
57.1.1 Detailed description of functions 892
57.2 PWR Firmware driver defines 897
57.2.1 PWR 897
58 LL RCC Generic Driver 899
58.1 RCC Firmware driver registers structures 899
58.1.1 LL_RCC_ClocksTypeDef 899
58.2 RCC Firmware driver API description 899
58.2.1 Detailed description of functions 899
58.3 RCC Firmware driver defines 921
58.3.1 RCC 921
59 LL RTC Generic Driver 927

59.1 RTC Firmware driver registers structures 927
59.1.1 LL_RTC_InitTypeDef 927
59.1.2 LL_RTC_TimeTypeDef 927
59.1.3 LL_RTC_AlarmTypeDef 927
59.2 RTC Firmware driver API description 928
59.2.1 Detailed description of functions 928
59.3 RTC Firmware driver defines 944
59.3.1 RTC 944
60 LL SPI Generic Driver 947
60.1 SPI Firmware driver registers structures 947
60.1.1 LL_SPI_InitTypeDef 947
60.2 SPI Firmware driver API description 948
60.2.1 Detailed description of functions 948
60.3 SPI Firmware driver defines 965
60.3.1 SPI 965
61 LL SYSTEM Generic Driver 967
61.1 SYSTEM Firmware driver API description 967
61.1.1 Detailed description of functions 967
61.2 SYSTEM Firmware driver defines 975
61.2.1 SYSTEM 975
62 LL TIM Generic Driver 977
62.1 TIM Firmware driver registers structures 977
62.1.1 LL_TIM_InitTypeDef 977
62.1.2 LL_TIM_OC_InitTypeDef 977
62.1.3 LL_TIM_IC_InitTypeDef 978
62.1.4 LL_TIM_ENCODER_InitTypeDef 979
62.1.5 LL_TIM_HALLSENSOR_InitTypeDef 980
62.1.6 LL_TIM_BDTR_InitTypeDef 980
62.2 TIM Firmware driver API description 981
62.2.1 Detailed description of functions 981
62.3 TIM Firmware driver defines 1043
62.3.1 TIM 1043
63 LL USART Generic Driver 1054
63.1 USART Firmware driver registers structures 1054
63.1.1 LL_USART_InitTypeDef 1054
63.1.2 LL_USART_ClockInitTypeDef 1054

63.2 USART Firmware driver API description 1055
63.2.1 Detailed description of functions 1055
63.3 USART Firmware driver defines 1099
63.3.1 USART 1099
64 LL UTILS Generic Driver 1103
64.1 UTILS Firmware driver registers structures 1103
64.1.1 LL_UTILS_PLLInitTypeDef 1103
64.1.2 LL_UTILS_ClkInitTypeDef 1103
64.2 UTILS Firmware driver API description 1103
64.2.1 System Configuration functions 1103
64.2.2 Detailed description of functions 1104
64.3 UTILS Firmware driver defines 1106
64.3.1 UTILS 1106
65 LL WWDG Generic Driver 1108
65.1 WWDG Firmware driver API description 1108
65.1.1 Detailed description of functions 1108
65.2 WWDG Firmware driver defines 1112
65.2.1 WWDG 1112
66 Correspondence between API registers and API low-layer driver functions 1113
66.1 ADC 1113
66.2 BUS 1117
66.3 CORTEX 1125
66.4 CRC 1126
66.5 DAC 1126
66.6 DMA 1128
66.7 EXTI 1131
66.8 GPIO 1131
66.9 I2C 1132
66.10 I2S 1136
66.11 IWDG 1137
66.12 PWR 1138
66.13 RCC 1138
66.14 RTC 1141
66.15 SPI 1142

66.16 SYSTEM 1144
66.17 TIM 1146
66.18 USART 1155
66.19 WWDG 1160
67 FAQs 1161
68 Revision history 1165

List of tables

Table 1: Acronyms and definitions....24
Table 2: HAL driver files....26
Table 3: User-application files....27
Table 4: API classification....32
Table 5: List of devices supported by HAL drivers....33
Table 6: HAL API naming rules....38
Table 7: Macros handling interrupts and specific clock configurations....39
Table 8: Callback functions....40
Table 9: HAL generic APIs....41
Table 10: HAL extension APIs....42
Table 11: Define statements used for HAL configuration....47
Table 12: Description of GPIO_InitTypeDef structure....49
Table 13: Description of EXTI configuration macros....51
Table 14: MSP functions....55
Table 15: Timeout values....59
Table 16: LL driver files....63
Table 17: Common peripheral initialization functions....66
Table 18: Optional peripheral initialization functions....67
Table 19: Specific Interrupt, DMA request and status flags management....69
Table 20: Available function formats....69
Table 21: Peripheral clock activation/deactivation management....69
Table 22: Peripheral activation/deactivation management....70
Table 23: Peripheral configuration management....70
Table 24: Peripheral register management....70
Table 25: Correspondence between ADC registers and ADC low-layer driver functions....1113
Table 26: Correspondence between BUS registers and BUS low-layer driver functions....1117
Table 27: Correspondence between CORTEX registers and CORTEX low-layer driver functions....1125
Table 28: Correspondence between CRC registers and CRC low-layer driver functions....1126
Table 29: Correspondence between DAC registers and DAC low-layer driver functions....1126
Table 30: Correspondence between DMA registers and DMA low-layer driver functions....1128
Table 31: Correspondence between EXTI registers and EXTI low-layer driver functions....1131
Table 32: Correspondence between GPIO registers and GPIO low-layer driver functions....1131
Table 33: Correspondence between I2C registers and I2C low-layer driver functions....1132
Table 34: Correspondence between I2S registers and I2S low-layer driver functions....1136
Table 35: Correspondence between IWDG registers and IWDG low-layer driver functions....1137
Table 36: Correspondence between PWR registers and PWR low-layer driver functions....1138
Table 37: Correspondence between RCC registers and RCC low-layer driver functions....1138
Table 38: Correspondence between RTC registers and RTC low-layer driver functions....1141
Table 39: Correspondence between SPI registers and SPI low-layer driver functions....1142
Table 40: Correspondence between SYSTEM registers and SYSTEM low-layer driver functions....1144
Table 41: Correspondence between TIM registers and TIM low-layer driver functions....1146
Table 42: Correspondence between USART registers and USART low-layer driver functions....1155
Table 43: Correspondence between WWDG registers and WWDG low-layer driver functions....1160
Table 44: Document revision history....1165

List of figures

Figure 1: Example of project template 29
Figure 2: Adding device-specific functions 43
Figure 3: Adding family-specific functions 43
Figure 4: Adding new peripherals 44
Figure 5: Updating existing APIs 44
Figure 6: File inclusion model 45
Figure 7: HAL driver model 53
Figure 8: Low Layer driver folders 64
Figure 9: Low Layer driver CMSIS files 65

1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
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
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 Controler
MSP	MCU Specific Package
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface

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

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode.
- Support of peripheral multi-instance 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 driver files

File	Description
stm32f1xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, …*
stm32f1xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example: stm32f1xx_hal_adc.h, stm32f1xx_hal_irda.h, …*

File	Description
stm32f1xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example: stm32f1 xx_hal_adc_ex.c, stm32f1xx_hal_dma_ex.c, …*
stm32f1xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f1xx_hal_adc_ex.h, stm32f1xx_hal_dma_ex.h, …*
stm32f1xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32f1xx_hal.h	xx_hal.c header file
stm32f1xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f1xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

2.1.2 User-application files

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

Table 3: User-application files

File	Description
system_stm32f1xx.c	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows to relocate the vector table in internal SRAM.
startup_stm32f1xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32f1xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
stm32f1xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
stm32f1xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32f1xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • the call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

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

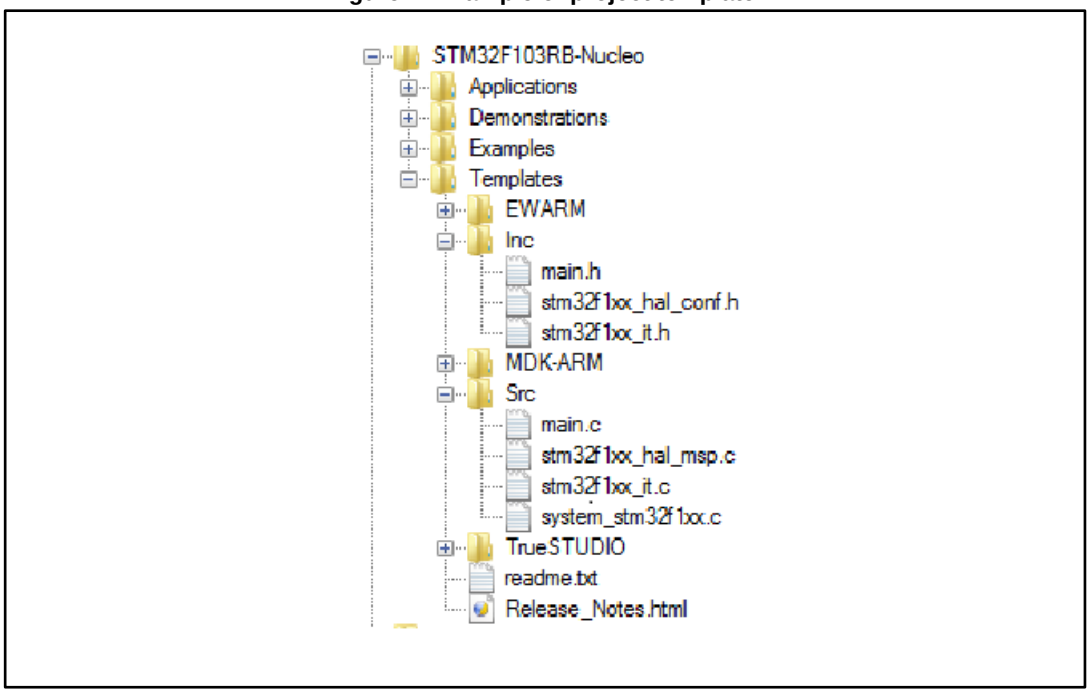
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- It contains the 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 configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized
  - SysTick ISR implemented for HAL_Delay()
  - System clock configured with the maximum frequency of the device



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

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

2.2.1 Peripheral handle structures

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

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

The peripheral handle is used for the following purposes:

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

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART InitTypeDef Init; /* Usart communication parameters */
uint8 t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
```

```
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

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

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

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

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

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

2.2.2 Initialization and configuration structure

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

```
typedef struct
{
uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
in a frame.*/
uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
uint32_t Parity; /*!< Specifies the parity mode. */
uint32_t Mode; /*!< Specifies wether the Receive or Transmit mode is enabled or
disabled.*/
uint32_t HwFlowCtl; /*!< Specifies wether the hardware flow control mode is enabled
or disabled.*/
```

```
uint32_t OverSampling; /*!< Specifies wether the Over sampling 8 is enabled or
disabled,
to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```



The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL ADC ConfigChannel (ADC HandleTypeDef* hadc, ADC ChannelConfTypeDef*
sConfig)
```

2.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

2.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:**common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver 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 (STM32F101xG) || defined (STM32F103x6)|| defined (STM32F103xB) ||
defined (STM32F105xC) || defined (STM32F107xC) || defined (STM32F103xE) || defined
(STM32F103xG)
/* ADC multimode */
HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA(ADC_HandleTypeDef *hadc, uint32_t
*pData, uint32 tLength);
HAL StatusTypeDef HAL ADCEx MultiModeStop DMA(ADC HandleTypeDef *hadc);
#endif
/* STM32F101xG || defined STM32F103x6 || defined STM32F103xB || defined STM32F105xC
|| defined STM32F107xC || defined STM32F103xE || defined STM32F103xG */
```



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: API 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

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_hal.c stm32f1xx_hal.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_adc.c stm32f1xx_hal_adc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_adc_ex.c stm32f1xx_hal_adc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_can.c stm32f1xx_hal_can.h	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_cec.c stm32f1xx_hal_cec.h	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No
stm32f1xx_hal_cortex.c stm32f1xx_hal_cortex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_crc.c stm32f1xx_hal_crc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_dac.c stm32f1xx_hal_dac.h	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_dac_ex.c stm32f1xx_hal_dac_ex.h	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_dma.c stm32f1xx_hal_dma.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_dma_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_eth.c stm32f1xx_hal_eth.h	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes
stm32f1xx_hal_flash.c stm32f1xx_hal_flash.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_flash_ex.c stm32f1xx_hal_flash_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_hal_gpio.c stm32f1xx_hal_gpio.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_gpio_ex.c stm32f1xx_hal_gpio_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_hcd.c stm32f1xx_hal_hcd.h	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes
stm32f1xx_hal_i2c.c stm32f1xx_hal_i2c.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_i2s.c stm32f1xx_hal_i2s.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_irda.c stm32f1xx_hal_irda.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_iwdg.c stm32f1xx_hal_iwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_msp_template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32f1xx_hal_nand.c stm32f1xx_hal_nand.h	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_nor.c stm32f1xx_hal_nor.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_pccard.c stm32f1xx_hal_pccard.h	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_pcd.c stm32f1xx_hal_pcd.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_pcd_ex.c stm32f1xx_hal_pcd_ex.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_hal_rcc.c stm32f1xx_hal_rcc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rcc_ex.c stm32f1xx_hal_rcc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rtc.c stm32f1xx_hal_rtc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rtc_ex.c stm32f1xx_hal_rtc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_sd.c stm32f1xx_hal_sd.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_smartcard.c stm32f1xx_hal_smartcard.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_spi.c stm32f1xx_hal_spi.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_sram.c stm32f1xx_hal_sram.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_tim.c stm32f1xx_hal_tim.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_tim_ex.c stm32f1xx_hal_tim_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_uart.c stm32f1xx_hal_uart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_usart.c stm32f1xx_hal_usart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_wwdg.c stm32f1xx_hal_wwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_ll_fsmc.c stm32f1xx_ll_fsmc.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_ll_sdmmc.c stm32f1xx_ll_sdmmc.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No
stm32f1xx_ll_usb.c stm32f1xx_ll_usb.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_adc.h stm32f1xx_ll_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_bus.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_cortex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_crc.h stm32f1xx_ll_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_dac.h stm32f1xx_ll_dac.c	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_ll_dma.h stm32f1xx_ll_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_exti.h stm32f1xx_ll_exti.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_gpio.h stm32f1xx_ll_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_i2c.h stm32f1xx_ll_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_iwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_pwr.h stm32f1xx_ll_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_rcc.h stm32f1xx_ll_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_ll_rtc.h stm32f1xx_ll_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_spi.h stm32f1xx_ll_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_system.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_tim.h stm32f1xx_ll_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_usart.h stm32f1xx_ll_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_utils.h stm32f1xx_ll_utils.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_wwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.5 HAL driver 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	*stm32f1xx_hal_ppp (c/h)*	*stm32f1xx_hal_ppp_ex (c/h)*	*stm32f1xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function HAL_PPP_FeatureFunction_ MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_ MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_ MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32F1 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in stm32f1xxx.h header file. stm32f1xxx.h corresponds to stm32f100xb.h, stm32f100xe.h, stm32f101x6.h, stm32f101xb.h, stm32f101xe.h, stm32f101xg.h, stm32f102x6.h, stm32f102xb.h, stm32f103x6.h, stm32f103xb.h, stm32f103xe.h, stm32f103xg.h, stm32f105xc.h and stm32f107xc.h.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (e.g. ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (e.g. ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (e.g. HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named PPP_DeInit, e.g. TIM_DeInit.

- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA ()*.
- The **Feature** prefix should refer to the new feature.
  Example: *HAL_ADC_Start*() refers to the injection mode

2.5.2 HAL general naming rules

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

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL StatusTypeDef HAL GPIO Init (GPIO TypeDef* GPIOx, GPIO InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

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

Table 7: Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the stm32f1xx_hal_cortex.c file.

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

```
HAL PPP Init(PPP HandleTypeDef)
if(hppp == NULL)
{
return HAL_ERROR;
}
```

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

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

  - Pseudo-code macro (multiple instructions macro):

```
#define   HAL LINKDMA(  HANDLE  ,   PPP DMA FIELD ,   DMA HANDLE ) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32f1xx_it.c
- User callback functions.

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

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8: Callback functions

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

2.6 HAL generic APIs

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

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

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

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

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

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

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

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

Table 9: HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests

Function group	Common API name	Description
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 HAL extension model overview

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

The extension model consists of an additional file, stm32f1xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32f1xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration

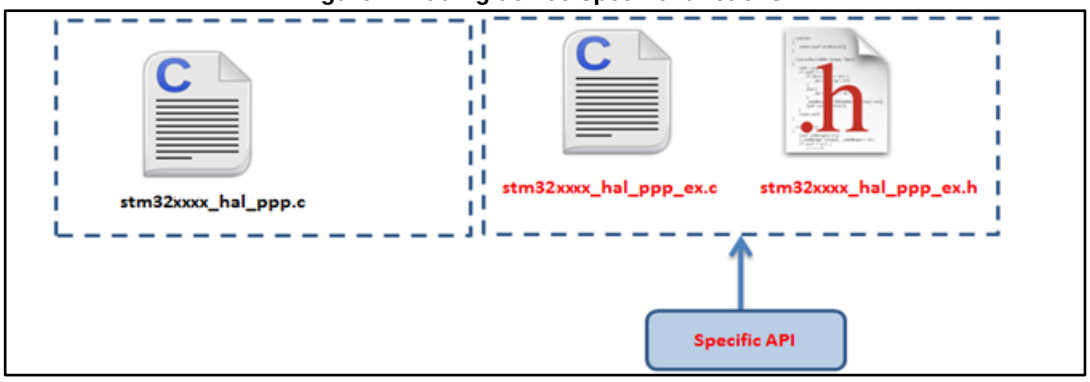
2.7.2 HAL extension model cases

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

Case 1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32f1xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2: Adding device-specific functions



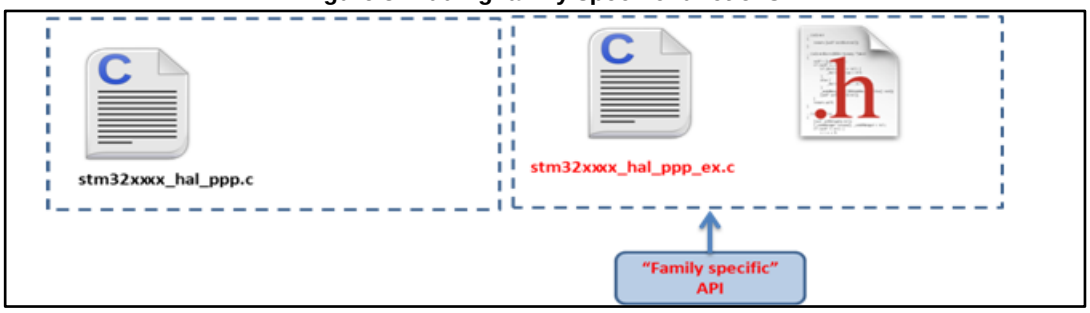
Example: stm32f1xx_hal_adc_ex.c/h

```
#if defined(STM32F101xG) || defined (STM32F103x6) || defined (STM32F103xB) ||
defined (STM32F105xC) ||
defined (STM32F107xC) || defined (STM32F103xE) || defined(STM32F103xG)
/* ADC multimode */
HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA(ADC_HandleTypeDef *hadc, uint32_t
*pData, uint32_t Length);
HAL StatusTypeDef HAL ADCEx MultiModeStop DMA(ADC HandleTypeDef *hadc);
#endif /* STM32F101xG || defined STM32F103x6 || defined STM32F103xB || defined
STM32F105xC ||
defined STM32F107xC || defined STM32F103xE || defined STM32F103xG */
```

Case 2: 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

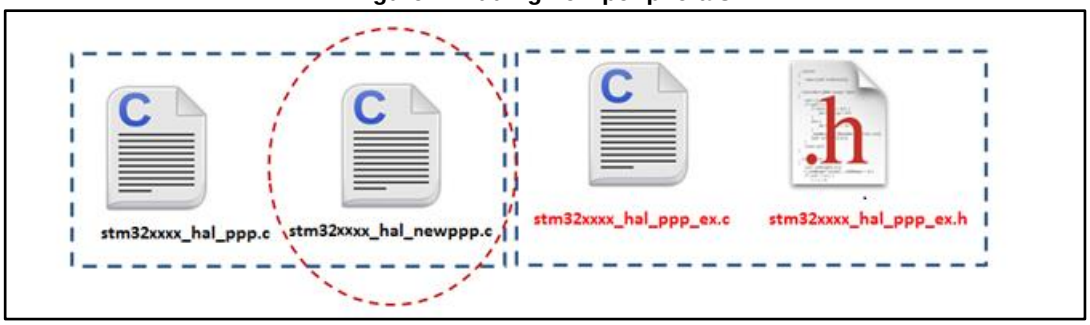


Case 3: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in stm32f1xx_hal_newppp.c. However the inclusion of this file is selected in the stm32lxx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

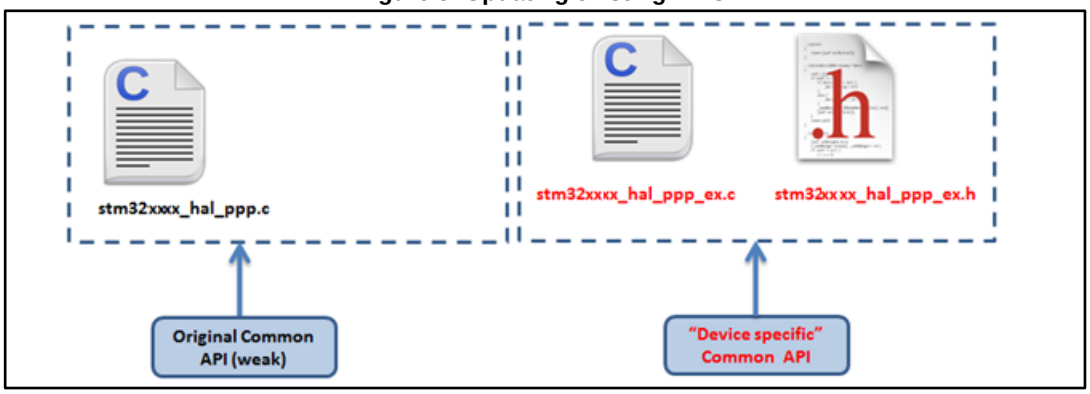


Example: stm32f1xx_hal_lcd.c/h

Case 4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32f1xx_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



Case 5: 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.

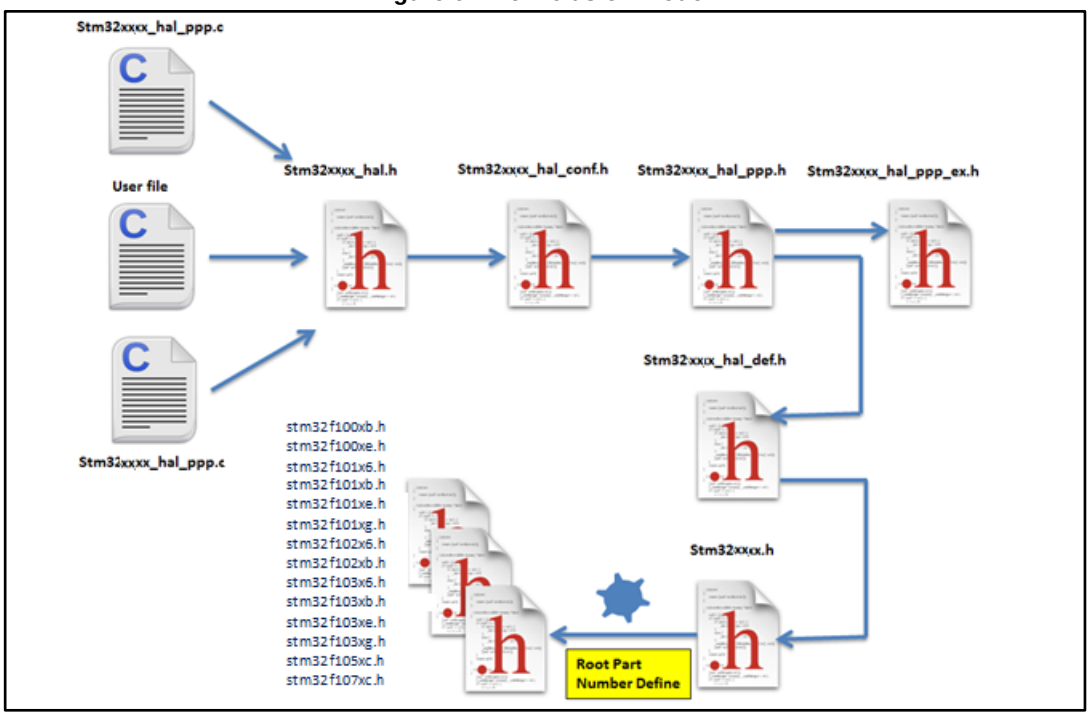
Example:

```
#if defined (STM32F100xB)
typedef struct
{
(…)
}PPP_InitTypeDef;
#endif /* STM32F100xB */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32f1xx_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 stm32f1xx 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 *stm32f1xx_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 stm32f1xx_hal_def.h file calls thestm32f1xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:

  - Declarations of peripheral registers and bits definition.
  - Macros to access peripheral registers hardware (Write register, Read register…etc.).
- **Common macros**
  - Macros defining NULL and HAL_MAX_DELAY

```
#ifndef NULL
#define NULL 0
#endif
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  - Macro linking a PPP peripheral to a DMA structure pointer: *__HAL_LINKDMA();*

```
#define   HAL LINKDMA(  HANDLE  ,   PPP DMA FIELD ,   DMA HANDLE ) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

2.10 HAL configuration

The configuration file*, stm32f1xx_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 on STM3210C-EVAL, otherwise 8 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	8 000 000 Hz
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 stm32f1xx_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 stm32f1xx_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 od 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

stm32f1xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

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

A set of macros are defined in stm32f1xx_hal_rcc.h and stm32f1xx_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), the 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 stm32f1xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 12: Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input floating – GPIO_MODE_OUTPUT_PP : Output push-pull – GPIO_MODE_OUTPUT_OD : Output open drain – GPIO_MODE_AF_PP : Alternate function push-pull – GPIO_MODE_AF_OD : Alternate function open drain – GPIO_MODE_ANALOG : Analog mode • External Interrupt mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_HIGH

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, stm32f1xx_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
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_GENERATE_SWIT()	Generates a software interrupt for a given EXTI line. Example: __HAL_PWR_PVD_EXTI_ GENERATE_SWIT ()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_ EVENT()	Enable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_ EVENT()	Disable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_ EDGE()	Configure an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_ EDGE()	Enable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_ EDGE()	Disable an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_ EDGE()	Disable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_ FALLING_EDGE()	Enable an EXTI Interrupt or Event on Rising/Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_ FALLING_EDGE()	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 stm32f1xx_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 confgured. In this case the DMA interrupt is configured.
  d. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
  e. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: 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”).

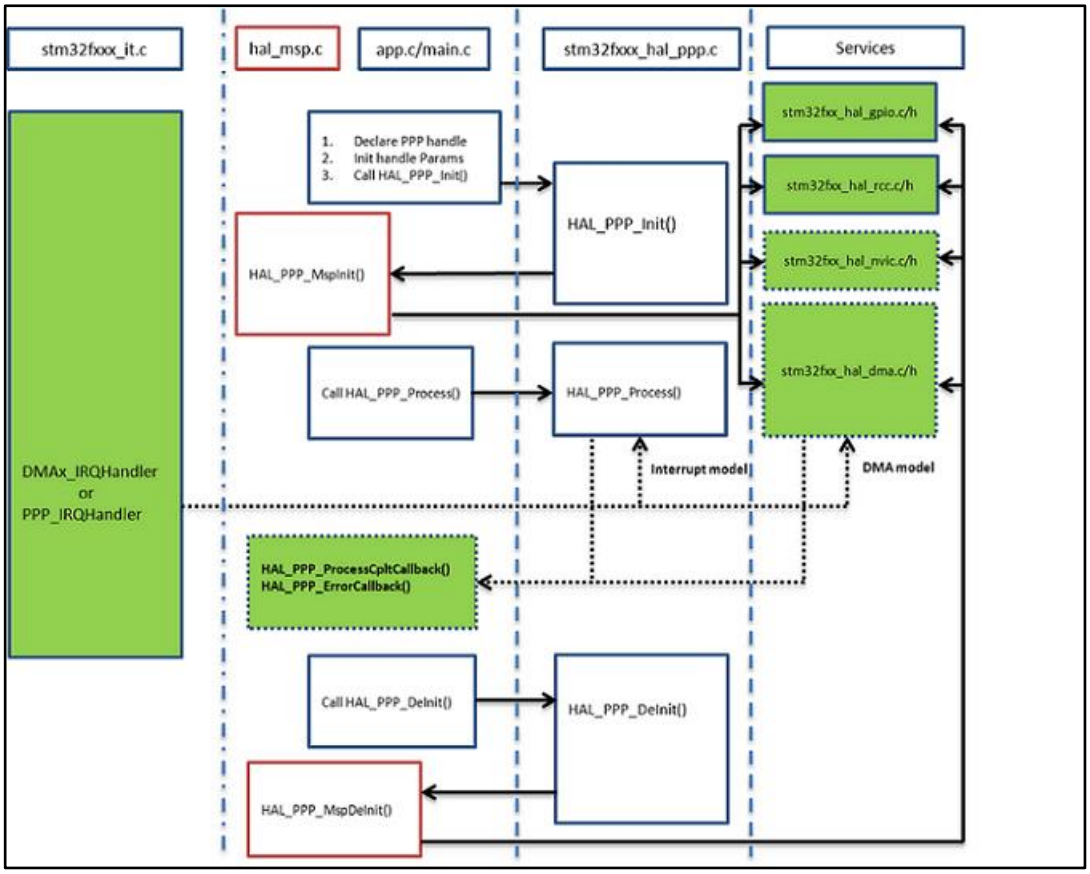
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

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 stm32f1xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in the HAL drivers.
- HAL_DeInit()
  - resets all peripherals
  - calls function HAL_MspDeInit() which a is user callback function to do system level De-Initalizations.
- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

2.12.2.2 System clock initialization

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

```
void SystemClock Config(void)
{
RCC_ClkInitTypeDef clkinitstruct = {0};
RCC_OscInitTypeDef oscinitstruct = {0};
/* Configure PLLs------------------------------------------------------*/
/* PLL2 configuration: PLL2CLK=(HSE/HSEPrediv2Value)*PLL2MUL=(25/5)*8=40 MHz */
/* PREDIV1 configuration: PREDIV1CLK = PLL2CLK / HSEPredivValue = 40 / 5 = 8 MHz */
/* PLL configuration: PLLCLK = PREDIV1CLK * PLLMUL = 8 * 9 = 72 MHz */
/* Enable HSE Oscillator and activate PLL with HSE as source */
oscinitstruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
oscinitstruct.HSEState =  RCC_HSE_ON;
oscinitstruct.HSEPredivValue =  RCC HSE PREDIV DIV5;
oscinitstruct.Prediv1Source = RCC_PREDIV1_SOURCE_PLL2;
oscinitstruct.PLL.PLLState = RCC_PLL_ON;
oscinitstruct.PLL.PLLSource =  RCC_PLLSOURCE_HSE;
oscinitstruct.PLL.PLLMUL = RCC PLL MUL9;
oscinitstruct.PLL2.PLL2State = RCC PLL2 ON;
oscinitstruct.PLL2.PLL2MUL =  RCC PLL2 MUL8;
oscinitstruct.PLL2.HSEPrediv2Value = RCC_HSE_PREDIV2_DIV5;
if  (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
```

```
{ /* Initialization Error */
while(1);

}
/* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2  clocks
dividers   */
clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK I  RCC_CLOCKTYPE_HCLK I
RCC_CLOCKTYPE_PCLK1 I RCC_CLOCKTYPE_PCLK2);
clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
clkinitstruct.AHBCLKDivider = RCC SYSCLK DIV1;
clkinitstruct.APB2CLKDivider =  RCC HCLK DIV1;
clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;
if  (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)
{ /*  Initialization Error */
 while(1);
}
}
```

2.12.2.3 HAL MSP initialization process

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit().*

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

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

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

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

The HAL MSP callback is implemented inside the *stm32f1xx_hal_msp.c* file in the user folders. An *stm32f1xx_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.

stm32f1xx_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

Routine	Description
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_ProcessCpltCallback ()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: the callback relative to the process Error.

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32f1xx_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)
{
}
```

*stm32f1xx_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 *stm32f1xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART STOPBITS 1;
UartHandle.Init.Parity = UART PARITY NONE;
UartHandle.Init.HwFlowCtl = UART HWCONTROL NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL USART MspInit (UART HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
  HAL LINKDMA(UartHandle, DMA Handle tx, hdma tx);
  HAL LINKDMA(UartHandle, DMA Handle rx, hdma rx);
(…)
}
```

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

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART DATABITS 8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART HWCONTROL NONE;
UartHandle.Init.Mode = UART MODE TX RX; UartHandle.Init.Instance = USART1;
HAL UART Init(&UartHandle);
HAL UART Send DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL UART TxCpltCallback(UART HandleTypeDef *phuart)
{
}
void HAL UART TxErrorCallback(UART HandleTypeDef *phuart)
{
}
```

stm32f1xx_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 stm32f1xx_hal_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL StatusTypeDef HAL PPP Process(PPP HandleTypeDef)
{
(…)
timeout = HAL_GetTick() + LOCAL_PROCESS_TIMEOUT;
(…)
while(ProcessOngoing)
{
(…)
if(HAL_GetTick() >= timeout)
{
/* Process unlocked */
  HAL UNLOCK(hppp);
hppp->State= HAL PPP STATE TIMEOUT;
return HAL_PPP_STATE_TIMEOUT;
}
```

```
}
(…)
}
```

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

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(…)
timeout = HAL GetTick() + Timeout;
(…)
while(ProcessOngoing)
{
(…)
if(Timeout != HAL MAX DELAY)
{
if(HAL GetTick() >= timeout)
{
/* Process unlocked */
  HAL UNLOCK(hppp);
hppp->State= HAL PPP STATE TIMEOUT;
return hppp->State;
}
}
(…)
}
```

2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

```
HAL StatusTypeDef HAL PPP Process(PPP HandleTypeDef* hppp, uint32 t *pdata, uint32
Size)
{
if ((pData == NULL ) || (Size == 0))
{
return HAL ERROR;
}
}
```

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

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

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

```
while (Process ongoing)
{
timeout = HAL GetTick() + Timeout; while (data processing is running)
{
if(timeout) { return HAL TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to allow retrieving the origin of the error.

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

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
  IO HAL PPP StateTypeDef State; /* PPP state */
  IO HAL PPP ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

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

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

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

```
void HAL PPP ProcessCpltCallback(PPP HandleTypeDef *hspi)
{
ErrorCode = HAL PPP GetError (hppp); /* retreive error code */
}
```

2.12.4.3 Run-time checking

The HAL implements run-time failure detection by checking the input values of all HAL driver functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver 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 *stm32f1xx_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 stm32f1xx_hal_conf.h:

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

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

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



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

3 Overview of Low Layer drivers

The Low Layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as FSMC, USB or SDMMC).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features.

The Low Layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

3.1 Low Layer files

The Low Layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some System and Cortex related features.

Table 16: LL driver files

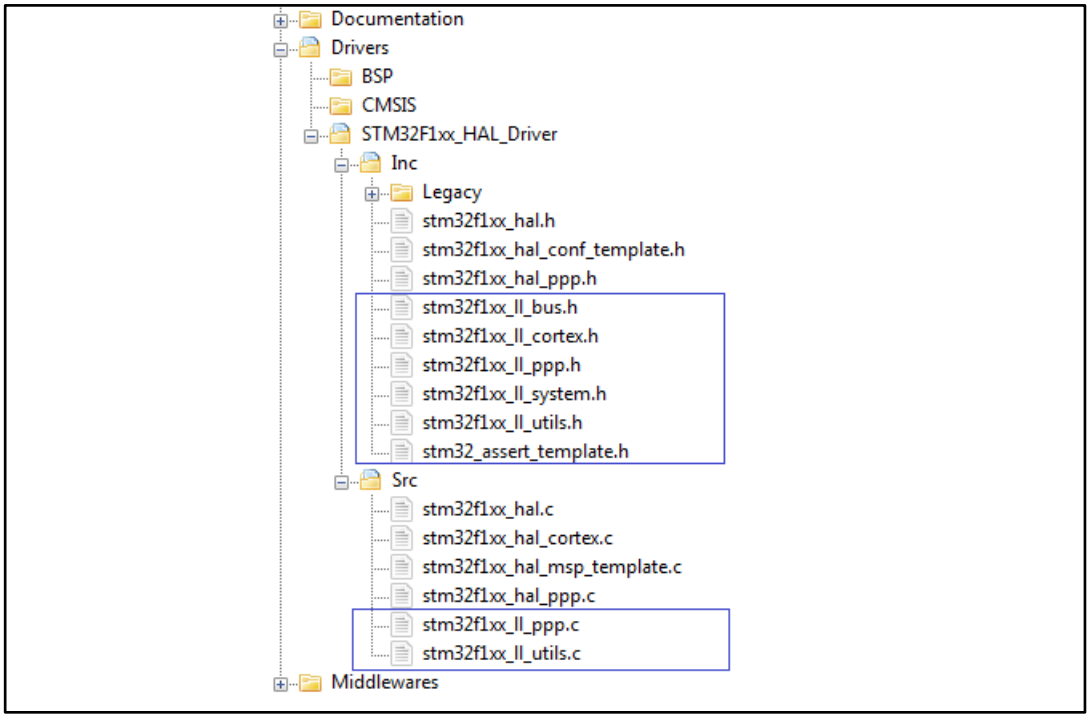
File	Description
stm32f1xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32f1xx_ll_ppp.h/.c	stm32f1xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32f1xx_ll_ppp.h file. The Low Layer PPP driver is a standalone module. To use it, the application must include it in the xx_ll_ppp.h file.
stm32f1xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (LL_SYSTICK_xxxxx, LL_LPM_xxxxx "Low Power Mode" ...)
stm32f1xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
*stm32f1*xx_ll_system.h	System related operations (LL_SYSCFG_xxx, LL_DBGMCU_xxx, LL_FLASH_xxx and LL_VREFBUF_xxx)

File	Description
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

There is no configuration file for the LL drivers.

The Low Layer files are located in the same HAL driver folder.

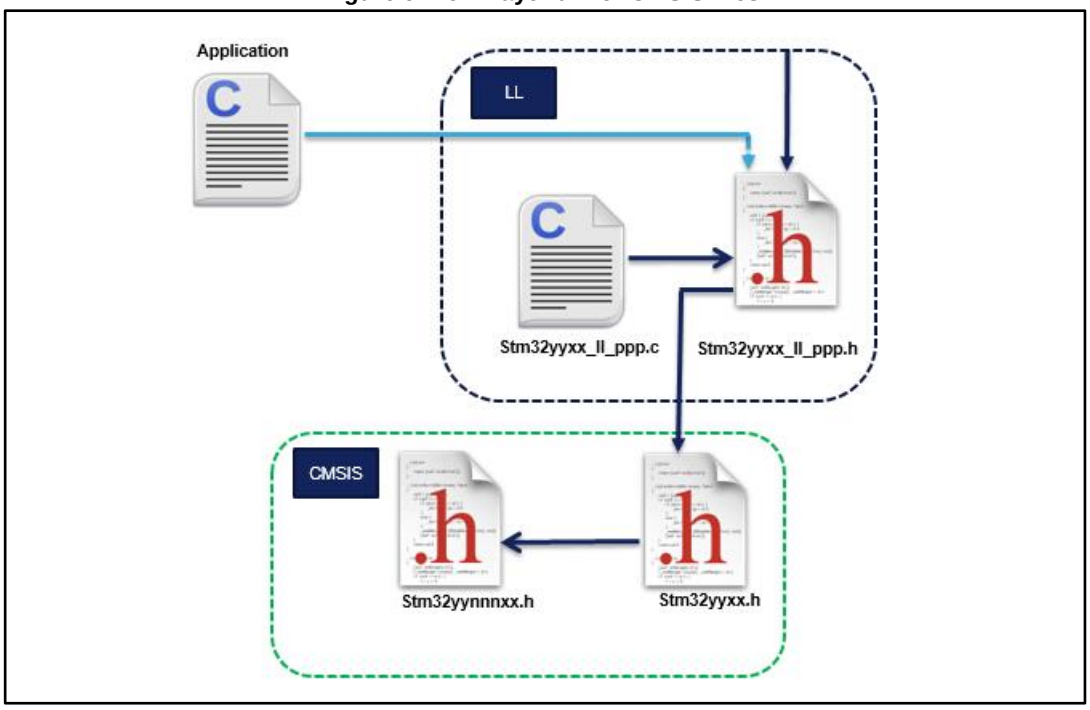
Figure 8: Low Layer driver folders



In general, Low Layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 9: Low Layer driver CMSIS files



Application files have to include only the used Low Layer driver header files.

3.2 Overview of Low Layer APIs and naming rules

3.2.1 Peripheral initialization functions

The LL drivers offer three set of initialization functions. They are defined in stm32f1xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral de-initialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 17: Common peripheral initialization functions

Functions	Return Type	Parameters	Description
LL_PPP_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to *Table 18: "Optional peripheral initialization functions"*).

Table 18: Optional peripheral initialization functions

Functions	Return Type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*Error Status*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)
LL_PPP_CommonInit	*Error Status*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)

Functions	Return Type	Parameters	Examples
LL_PPP_Common StructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each PPP_CommonInitStruct member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*Error Status*	• *PPP_TypeDef* PPPx* • *LL_PPP_ClockInit TypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

3.2.1.1 Run-time checking

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL driver functions. For more details please refer to *Section 2.12.4.3: "Run-time checking"*.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use run-time checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when run-time checking is enabled.
2. Include stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.



Run-time checking is not available for LL inline functions.

3.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending to the action category:

- **Specific Interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 19: Specific Interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 20: Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*



BITNAME refers to the peripheral register bit name as described in the product line reference manual.

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 21: Peripheral clock activation/deactivation management

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA|LL_AHB2_GRP1_PERIPH_GPIOB)* • *by LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

'x' corresponds to the group index and refers to the index of the modified register on a given bus.

- ***Peripheral activation/deactivation management***: Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 22: Peripheral activation/deactivation management

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management***: Set/get a peripheral configuration settings

Table 23: Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management***: Write/read the content of a register/retrun DMA relative register address

Table 24: Peripheral register management

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

The Propriety is a variable used to identify the DMA transfer direction or the data register type.

4 Cohabiting of HAL and LL

The Low Layer is designed to be used in standalone mode or combined with the HAL. It cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the Low Layer might overwrite some registers which content is mirrored in the HAL handles.

4.1 Low Layer driver used in standalone mode

The Low Layer APIs can be used without calling the HAL driver services. This is done by simply including stm32f1xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case the HAL drivers associated to the used peripheral can be removed from the workspace. However the STM32CubeF1 framework should be used in the same way as in the HAL drivers case which means that System file, startup file and CMSIS should always be used.



When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

4.2 Mixed use of Low Layer APIs and HAL drivers

In this case the Low Layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of Low Layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the Low Layer services can be used together for the same peripheral instance.
- The Low Layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, flash and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within stm32f1 firmware package (refer to Examples_MIX projects).



1. When the HAL Init/DeInit APIs are not used and are replaced by the Low Layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.
2. When process APIs are not used and the corresponding function is performed through the Low Layer APIs, the callbacks are not called and post processing or error management should be done by the user application.
3. When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.

5 HAL System Driver

5.1 HAL Firmware driver API description

5.1.1 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

5.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***
- ***HAL_InitTick()***

5.1.3 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_GetUID()***

5.1.4 Detailed description of functions

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.
Return values	• **HAL:** status
Notes	• 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.

HAL_DeInit

Function name	**HAL_StatusTypeDef HAL_DeInit (void)**
Function description	This function de-Initializes common part of the HAL and stops the source of time base.
Return values	• **HAL:** status
Notes	• This function is optional.

HAL_MspInit

Function name	**void HAL_MspInit (void)**
Function description	Initializes the MSP.
Return values	• **None:**

HAL_MspDeInit

Function name	**void HAL_MspDeInit (void)**
Function description	DeInitializes the MSP.
Return values	• **None:**

HAL_InitTick

Function name	**HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)**
Function description	This function configures the source of the time base.
Parameters	• **TickPriority:** Tick interrupt priority.
Return values	• **HAL:** status
Notes	• This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig(). • In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name	**void HAL_IncTick (void)**
Function description	This function is called to increment a global variable "uwTick" used as application time base.
Return values	• **None:**
Notes	• In the default implementation, this variable is incremented each 1ms in Systick ISR. • This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name	**void HAL_Delay (__IO uint32_t Delay)**
Function description	This function provides minimum delay (in milliseconds) based on variable incremented.
Parameters	• **Delay:** specifies the delay time length, in milliseconds.
Return values	• **None:**
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented. • This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name	**uint32_t HAL_GetTick (void)**
Function description	Provides a tick value in millisecond.
Return values	• **tick:** value
Notes	• This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_SuspendTick

Function name	**void HAL_SuspendTick (void)**
Function description	Suspend Tick increment.
Return values	• **None:**
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the 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.

HAL_ResumeTick

Function name	**void HAL_ResumeTick (void)**
Function description	Resume Tick increment.
Return values	• **None:**
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the 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.

HAL_GetHalVersion

Function name	**uint32_t HAL_GetHalVersion (void)**
Function description	Returns the HAL revision.
Return values	• **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name	**uint32_t HAL_GetREVID (void)**
Function description	Returns the device revision identifier.
Return values	• **Device:** revision identifier

HAL_GetDEVID

Function name	**uint32_t HAL_GetDEVID (void)**

Function description Returns the device identifier.

Return values
- **Device:** identifier

HAL_DBGMCU_EnableDBGSleepMode

Function name **void HAL_DBGMCU_EnableDBGSleepMode (void)**

Function description Enable the Debug Module during SLEEP mode.

Return values
- **None:**

HAL_DBGMCU_DisableDBGSleepMode

Function name **void HAL_DBGMCU_DisableDBGSleepMode (void)**

Function description Disable the Debug Module during SLEEP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values
- **None:**

HAL_DBGMCU_EnableDBGStopMode

Function name **void HAL_DBGMCU_EnableDBGStopMode (void)**

Function description Enable the Debug Module during STOP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values
- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name **void HAL_DBGMCU_DisableDBGStopMode (void)**

Function description Disable the Debug Module during STOP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values
- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name **void HAL_DBGMCU_EnableDBGStandbyMode (void)**

Function description Enable the Debug Module during STANDBY mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers

DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values
- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name **void HAL_DBGMCU_DisableDBGStandbyMode (void)**

Function description Disable the Debug Module during STANDBY mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values
- **None:**

HAL_GetUID

Function name **void HAL_GetUID (uint32_t * UID)**

Function description Return the unique device identifier (UID based on 96 bits)

Parameters
- **UID:** pointer to 3 words array.

Return values
- **Device:** identifier

5.2 HAL Firmware driver defines

5.2.1 HAL

Freeze Unfreeze Peripherals in Debug mode

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM4

__HAL_DBGMCU_UNFREEZE_TIM4

__HAL_DBGMCU_FREEZE_TIM5

__HAL_DBGMCU_UNFREEZE_TIM5

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_TIM12

__HAL_DBGMCU_UNFREEZE_TIM12

__HAL_DBGMCU_FREEZE_TIM13

__HAL_DBGMCU_UNFREEZE_TIM13
__HAL_DBGMCU_FREEZE_TIM14
__HAL_DBGMCU_UNFREEZE_TIM14
__HAL_DBGMCU_FREEZE_WWDG
__HAL_DBGMCU_UNFREEZE_WWDG
__HAL_DBGMCU_FREEZE_IWDG
__HAL_DBGMCU_UNFREEZE_IWDG
__HAL_DBGMCU_FREEZE_I2C1_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT
__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT
__HAL_DBGMCU_FREEZE_CAN1
__HAL_DBGMCU_UNFREEZE_CAN1
__HAL_DBGMCU_FREEZE_TIM1
__HAL_DBGMCU_UNFREEZE_TIM1
__HAL_DBGMCU_FREEZE_TIM8
__HAL_DBGMCU_UNFREEZE_TIM8
__HAL_DBGMCU_FREEZE_TIM9
__HAL_DBGMCU_UNFREEZE_TIM9
__HAL_DBGMCU_FREEZE_TIM10
__HAL_DBGMCU_UNFREEZE_TIM10
__HAL_DBGMCU_FREEZE_TIM11
__HAL_DBGMCU_UNFREEZE_TIM11

6 HAL ADC Generic Driver

6.1 ADC Firmware driver registers structures

6.1.1 ADC_InitTypeDef

Data Fields

- ***uint32_t DataAlign***
- ***uint32_t ScanConvMode***
- ***uint32_t ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***uint32_t DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***

Field Documentation

- ***uint32_t ADC_InitTypeDef::DataAlign***
  Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset application): MSB on register bit 14 and LSB on register bit 3). This parameter can be a value of ***ADC_Data_align***
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  Configures the sequencer of regular and injected groups. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion'/'InjectedNbrOfConversion' and each channel rank). Scan direction is upward: from rank1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode***
  Note: For regular group, this parameter should be enabled in conversion either by polling (HAL_ADC_Start with Discontinuous mode and NbrOfDiscConversion=1) or by DMA (HAL_ADC_Start_DMA), but not by interruption (HAL_ADC_Start_IT): in scan mode, interruption is triggered only on the the last conversion of the sequence. All previous conversions would be overwritten by the last one. Injected group used with scan mode has not this constraint: each rank has its own result register, no data is overwritten.
- ***uint32_t ADC_InitTypeDef::ContinuousConvMode***
  Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
  Specifies the number of ranks that will be converted within the regular group sequencer. To use regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16.
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode***
  Specifies whether the conversions sequence of regular group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded.

Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
  Specifies the number of discontinuous conversions in which the main sequence of regular group (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  Selects the external event used to trigger the conversion start of regular group. If set to ADC_SOFTWARE_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of ***ADC_External_trigger_source_Regular***

6.1.2 ADC_ChannelConfTypeDef

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specifies the channel to configure into ADC regular group. This parameter can be a value of ***ADC_channels*** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Note: On STM32F1 devices with several ADC: Only ADC1 can access internal measurement channels (VrefInt/TempSensor) Note: On STM32F10xx8 and STM32F10xxB devices: A low-amplitude voltage glitch may be generated (on ADC input 0) on the PA0 pin, when the ADC is converting with injection trigger. It is advised to distribute the analog channels so that Channel 0 is configured as an injected channel. Refer to errata sheet of these devices for more details.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Specifies the rank in the regular group sequencer This parameter can be a value of ***ADC_regular_rank*** Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted)
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits). This parameter can be a value of ***ADC_sampling_times*** Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_temp (values rough order: 5us to 17.1us min).

6.1.3 ADC_AnalogWDGConfTypeDef

Data Fields

- ***uint32_t WatchdogMode***
- ***uint32_t Channel***

- ***uint32_t ITMode***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***
- ***uint32_t WatchdogNumber***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  Configures the ADC analog watchdog mode: single/all channels, regular/injected group. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Selects which ADC channel to monitor by analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel (parameter WatchdogMode) This parameter can be a value of ***ADC_channels***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::ITMode***
  Specifies whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  Configures the ADC analog watchdog High threshold value. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog High threshold value. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
  Reserved for future use, can be set to 0

6.1.4 ADC_HandleTypeDef

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***ADC_TypeDef* ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef ADC_HandleTypeDef::Init***
  ADC required parameters
- ***DMA_HandleTypeDef* ADC_HandleTypeDef::DMA_Handle***
  Pointer DMA Handler
- ***HAL_LockTypeDef ADC_HandleTypeDef::Lock***
  ADC locking object
- ***__IO uint32_t ADC_HandleTypeDef::State***
  ADC communication state (bitmap of ADC states)
- ***__IO uint32_t ADC_HandleTypeDef::ErrorCode***
  ADC Error code

6.2 ADC Firmware driver API description

6.2.1 ADC peripheral features

- 12-bit resolution

- Interrupt generation at the end of regular conversion, end of injected conversion, and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- ADC conversion of regular group and injected group.
- External trigger (timer or EXTI) for both regular and injected groups.
- DMA request generation for transfer of conversions data of regular group.
- Multimode Dual mode (available on devices with 2 ADCs or more).
- Configurable DMA data storage in Multimode Dual mode (available on devices with 2 DCs or more).
- Configurable delay between conversions in Dual interleaved mode (available on devices with 2 DCs or more).
- ADC calibration
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

6.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level. Caution: On STM32F1, ADC clock frequency max is 14MHz (refer to device datasheet). Therefore, ADC clock prescaler must be configured in function of ADC clock source frequency to remain below this maximum frequency.
   - One clock setting is mandatory: ADC clock (core clock, also possibly conversion clock).
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - RCC_PeriphCLKInitTypeDef PeriphClkInit;
     - __ADC1_CLK_ENABLE();
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
   - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

- Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, groups regular/injected, channels parameters

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

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().
2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   - ADC conversion by polling:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
     - Wait for ADC conversion completion using function HAL_ADC_PollForConversion() (or for injected group: HAL_ADCEx_InjectedPollForConversion())
     - Retrieve conversion results using function HAL_ADC_GetValue() (or for injected group: HAL_ADCEx_InjectedGetValue())
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   - ADC conversion by interruption:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program) (or for injected group: HAL_ADCEx_InjectedConvCpltCallback())
     - Retrieve conversion results using function HAL_ADC_GetValue() (or for injected group: HAL_ADCEx_InjectedGetValue())
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   - ADC conversion with transfer by DMA:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
     - Conversion results are automatically transferred by DMA into destination variable address.
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

  - For devices with several ADCs: ADC multimode conversion with transfer by DMA:
    - Activate the ADC peripheral (slave) and start conversions using function HAL_ADC_Start()
    - Activate the ADC peripheral (master) and start conversions using function HAL_ADCEx_MultiModeStart_DMA()
    - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
    - Conversion results are automatically transferred by DMA into destination variable address.
    - Stop conversion and disable the ADC peripheral (master) using function HAL_ADCEx_MultiModeStop_DMA()
    - Stop conversion and disable the ADC peripheral (slave) using function HAL_ADC_Stop_IT()



Callback functions must be implemented in user program:

- HAL_ADC_ErrorCallback()
- HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)
- HAL_ADC_ConvCpltCallback()
- HAL_ADC_ConvHalfCpltCallback
- HAL_ADCEx_InjectedConvCpltCallback()

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC
     - PeriphClkInit.AdcClockSelection = RCC_ADCPLLCLK2_OFF
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit)
2. ADC pins configuration
   - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   - Deinitialize the DMA using function HAL_DMA_Init().
   - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

6.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_ADC_Init()***
- ***HAL_ADC_DeInit()***
- ***HAL_ADC_MspInit()***
- ***HAL_ADC_MspDeInit()***

6.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular group.
- Stop conversion of regular group.
- Poll for conversion complete on regular group.
- Poll for conversion event.
- Get result of regular channel conversion.
- Start conversion of regular group and enable interruptions.
- Stop conversion of regular group and disable interruptions.
- Handle ADC interrupt request
- Start conversion of regular group and enable DMA transfer.
- Stop conversion of regular group and disable ADC DMA transfer.

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_Start_DMA()***
- ***HAL_ADC_Stop_DMA()***
- ***HAL_ADC_GetValue()***
- ***HAL_ADC_IRQHandler()***
- ***HAL_ADC_ConvCpltCallback()***
- ***HAL_ADC_ConvHalfCpltCallback()***
- ***HAL_ADC_LevelOutOfWindowCallback()***
- ***HAL_ADC_ErrorCallback()***

6.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- ***HAL_ADC_ConfigChannel()***
- ***HAL_ADC_AnalogWDGConfig()***

6.2.6 Peripheral State and Errors functions

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

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- ***HAL_ADC_GetState()***
- ***HAL_ADC_GetError()***

6.2.7 Detailed description of functions

HAL_ADC_Init

Function name **HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)**

Function description Initializes the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status

Notes
- As prerequisite, ADC clock must be configured at RCC top level (clock source APB2). See commented example code below that can be copied and uncommented into HAL_ADC_MspInit().
- Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".
- This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".

HAL_ADC_DeInit

Function name **HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)**

Function description Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status

HAL_ADC_MspInit

Function name **void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)**

Function description Initializes the ADC MSP.

Parameters
- **hadc:** ADC handle

Return values
- **None:**

HAL_ADC_MspDeInit

Function name **void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)**

Function description DeInitializes the ADC MSP.

Parameters
- **hadc:** ADC handle

Return values
- **None:**

HAL_ADC_Start

Function name **HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)**

Function description Enables ADC, starts conversion of regular group.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status

HAL_ADC_Stop

Function name **HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)**

Function description Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status.

Notes
- : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name **HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)**

Function description Wait for regular group conversion to be completed.

Parameters
- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values
- **HAL:** status

Notes
- This function cannot be used in a particular setup: ADC configured in DMA mode. In this case, DMA resets the flag EOC and polling cannot be performed on each conversion.
- On STM32F1 devices, limitation in case of sequencer enabled (several ranks selected): polling cannot be done on each conversion inside the sequence. In this case, polling is replaced by wait for maximum conversion time.

HAL_ADC_PollForEvent

Function name **HAL_StatusTypeDef HAL_ADC_PollForEvent**

(ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description: Poll for conversion event.

Parameters:
- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values:
  – ADC_AWD_EVENT: ADC Analog watchdog event.
- **Timeout:** Timeout value in millisecond.

Return values:
- **HAL:** status

HAL_ADC_Start_IT

Function name: **HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)**

Function description: Enables ADC, starts conversion of regular group with interruption.

HAL_ADC_Stop_IT

Function name: **HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)**

Function description: Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.

Parameters:
- **hadc:** ADC handle

Return values:
- **None:**

HAL_ADC_Start_DMA

Function name: **HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)**

Function description: Enables ADC, starts conversion of regular group and transfers result through DMA.

HAL_ADC_Stop_DMA

Function name: **HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)**

Function description: Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters:
- **hadc:** ADC handle

Return values:
- **HAL:** status.

Notes:
- : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.
- For devices with several ADCs: This function is for single-

ADC mode only. For multimode, use the dedicated MultimodeStop function.
- On STM32F1 devices, only ADC1 and ADC3 (ADC availability depending on devices) have DMA capability.

HAL_ADC_GetValue

Function name	**uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)**
Function description	Get ADC regular group conversion result.
Parameters	• **hadc:** ADC handle
Return values	• **ADC:** group regular conversion data
Notes	• Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion). • This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_IRQHandler

Function name	**void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)**
Function description	Handles ADC interrupt request.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_ConvCpltCallback

Function name	**void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Conversion complete callback in non blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_ConvHalfCpltCallback

Function name	**void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)**
Function description	Conversion DMA half-transfer callback in non blocking mode.
Parameters	• **hadc:** ADC handle
Return values	• **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name **void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)**

Function description Analog watchdog callback in non blocking mode.

Parameters
- **hadc:** ADC handle

Return values
- **None:**

HAL_ADC_ErrorCallback

Function name **void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)**

Function description ADC error callback in non blocking mode (ADC conversion with interruption or transfer by DMA)

Parameters
- **hadc:** ADC handle

Return values
- **None:**

HAL_ADC_ConfigChannel

Function name **HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)**

Function description Configures the the selected channel to be linked to the regular group.

Parameters
- **hadc:** ADC handle
- **sConfig:** Structure of ADC channel for regular group.

Return values
- **HAL:** status

Notes
- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be be disabled using function HAL_ADC_DeInit().
- Possibility to update parameters on the fly: This function initializes channel into regular group, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without reseting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name **HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)**

Function description Configures the analog watchdog.

Parameters
- **hadc:** ADC handle
- **AnalogWDGConfig:** Structure of ADC analog watchdog configuration

Return values
- **HAL:** status

Notes
- Analog watchdog thresholds can be modified while ADC conversion is on going. In this case, some constraints must be taken into account: the programmed threshold values are effective from the next ADC EOC (end of unitary conversion). Considering that registers write delay may happen due to bus activity, this might cause an uncertainty on the effective timing of the new programmed threshold values.

HAL_ADC_GetState

Function name **uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)**

Function description return the ADC state

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** state

HAL_ADC_GetError

Function name **uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)**

Function description Return the ADC error code.

Parameters
- **hadc:** ADC handle

Return values
- **ADC:** Error Code

ADC_Enable

Function name **HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)**

Function description Enable the selected ADC.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status.

Notes
- Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_ConversionStop_Disable

Function name **HAL_StatusTypeDef ADC_ConversionStop_Disable (ADC_HandleTypeDef * hadc)**

Function description Stop ADC conversion and disable the selected ADC.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status.

Notes
- Prerequisite condition to use this function: ADC conversions must be stopped to disable the ADC.

ADC_StabilizationTime

Function name **void ADC_StabilizationTime (uint32_t DelayUs)**

Function description

ADC_DMAConvCplt

Function name **void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)**

Function description DMA transfer complete callback.

Parameters
- **hdma:** pointer to DMA handle.

Return values
- **None:**

ADC_DMAHalfConvCplt

Function name **void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)**

Function description DMA half transfer complete callback.

Parameters
- **hdma:** pointer to DMA handle.

Return values
- **None:**

ADC_DMAError

Function name **void ADC_DMAError (DMA_HandleTypeDef * hdma)**

Function description DMA error callback.

Parameters
- **hdma:** pointer to DMA handle.

Return values
- **None:**

6.3 ADC Firmware driver defines

6.3.1 ADC

ADC analog watchdog mode

ADC_ANALOGWATCHDOG_NONE

ADC_ANALOGWATCHDOG_SINGLE_REG

ADC_ANALOGWATCHDOG_SINGLE_INJEC

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

ADC_ANALOGWATCHDOG_ALL_REG

ADC_ANALOGWATCHDOG_ALL_INJEC

ADC_ANALOGWATCHDOG_ALL_REGINJEC

ADC channels

ADC_CHANNEL_0

ADC_CHANNEL_1

ADC_CHANNEL_2

ADC_CHANNEL_3

ADC_CHANNEL_4

ADC_CHANNEL_5

ADC_CHANNEL_6

ADC_CHANNEL_7

ADC_CHANNEL_8

ADC_CHANNEL_9

ADC_CHANNEL_10

ADC_CHANNEL_11

ADC_CHANNEL_12

ADC_CHANNEL_13

ADC_CHANNEL_14

ADC_CHANNEL_15

ADC_CHANNEL_16

ADC_CHANNEL_17

ADC_CHANNEL_TEMPSENSOR

ADC_CHANNEL_VREFINT

ADC conversion cycles

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_1CYCLE5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_7CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_13CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_28CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_41CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_55CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_71CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_239CYCLES5

ADC conversion group

ADC_REGULAR_GROUP

ADC_INJECTED_GROUP

ADC_REGULAR_INJECTED_GROUP

ADC data alignment

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC Error Code

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

ADC Event type

ADC_AWD_EVENT	ADC Analog watchdog event
ADC_AWD1_EVENT	ADC Analog watchdog 1 event: Alternate naming for compatibility with other STM32 devices having several analog watchdogs

ADC Exported Macros

__HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

Notes:

- ADC enable requires a delay for ADC stabilization time (refer to device datasheet, parameter tSTAB) On STM32F1, if ADC is already enabled this macro trigs a conversion SW start on regular group.

__HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the following values:
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_AWD: ADC Analog watchdog interrupt source

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the following values:
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_AWD: ADC Analog watchdog interrupt source

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be any combination of the following values:
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_AWD: ADC Analog watchdog interrupt source

Return value:

- None

__HAL_ADC_GET_FLAG

Description:

- Get the selected ADC's flag status.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be any combination of the following values:
  - ADC_FLAG_STRT: ADC Regular group start flag
  - ADC_FLAG_JSTRT: ADC Injected group start flag
  - ADC_FLAG_EOC: ADC End of Regular conversion flag
  - ADC_FLAG_JEOC: ADC End of Injected conversion flag
  - ADC_FLAG_AWD: ADC Analog watchdog flag

Return value:

- None

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the ADC's pending flags.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be any combination of the following values:
  - ADC_FLAG_STRT: ADC Regular group start flag
  - ADC_FLAG_JSTRT: ADC Injected group start flag
  - ADC_FLAG_EOC: ADC End of Regular conversion flag
  - ADC_FLAG_JEOC: ADC End of Injected conversion flag
  - ADC_FLAG_AWD: ADC Analog watchdog flag

Return value:

- None

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC Exported Types

HAL_ADC_STATE_RESET	ADC not yet initialized or disabled
HAL_ADC_STATE_READY	ADC peripheral ready for use
HAL_ADC_STATE_BUSY_INTERNAL	ADC is busy to internal process (initialization, calibration)
HAL_ADC_STATE_TIMEOUT	TimeOut occurrence
HAL_ADC_STATE_ERROR_INTERNAL	Internal error occurrence
HAL_ADC_STATE_ERROR_CONFIG	Configuration error occurrence
HAL_ADC_STATE_ERROR_DMA	DMA error occurrence
HAL_ADC_STATE_REG_BUSY	A conversion on group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on, multimode ADC master control)
HAL_ADC_STATE_REG_EOC	Conversion data available on group regular
HAL_ADC_STATE_REG_OVR	Not available on STM32F1 device: Overrun occurrence
HAL_ADC_STATE_REG_EOSMP	Not available on STM32F1 device: End Of

	Sampling flag raised
HAL_ADC_STATE_INJ_BUSY	A conversion on group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on, multimode ADC master control)
HAL_ADC_STATE_INJ_EOC	Conversion data available on group injected
HAL_ADC_STATE_INJ_JQOVF	Not available on STM32F1 device: Injected queue overflow occurrence
HAL_ADC_STATE_AWD1	Out-of-window occurrence of analog watchdog 1
HAL_ADC_STATE_AWD2	Not available on STM32F1 device: Out-of-window occurrence of analog watchdog 2
HAL_ADC_STATE_AWD3	Not available on STM32F1 device: Out-of-window occurrence of analog watchdog 3
HAL_ADC_STATE_MULTIMODE_SLAVE	ADC in multimode slave state, controlled by another ADC master (

ADC external trigger enable for regular group

ADC_EXTERNALTRIGCONVEDGE_NONE

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC External trigger selection for regular group

ADC_EXTERNALTRIGCONV_T1_CC1	< List of external triggers with generic trigger name, independently of
ADC_EXTERNALTRIGCONV_T1_CC2	
ADC_EXTERNALTRIGCONV_T2_CC2	
ADC_EXTERNALTRIGCONV_T3_TRGO	
ADC_EXTERNALTRIGCONV_T4_CC4	
ADC_EXTERNALTRIGCONV_EXT_IT11	External triggers of regular group for ADC3 only
ADC_EXTERNALTRIGCONV_T2_CC3	
ADC_EXTERNALTRIGCONV_T3_CC1	
ADC_EXTERNALTRIGCONV_T5_CC1	
ADC_EXTERNALTRIGCONV_T5_CC3	
ADC_EXTERNALTRIGCONV_T8_CC1	
ADC_EXTERNALTRIGCONV_T1_CC3	< External triggers of regular group for all ADC instances Note: TIM8_TRGO is available on ADC1 and ADC2 only in high-density and
ADC_EXTERNALTRIGCONV_T8_TRGO	
ADC_SOFTWARE_START	

ADC flags definition

ADC_FLAG_STRT	ADC Regular group start flag

ADC_FLAG_JSTRT	ADC Injected group start flag
ADC_FLAG_EOC	ADC End of Regular conversion flag
ADC_FLAG_JEOC	ADC End of Injected conversion flag
ADC_FLAG_AWD	ADC Analog watchdog flag

ADC interrupts definition

ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_JEOC	ADC End of Injected Conversion interrupt source
ADC_IT_AWD	ADC Analog watchdog interrupt source

ADC range verification

IS_ADC_RANGE

ADC regular discontinuous mode number verification

IS_ADC_REGULAR_DISCONT_NUMBER

ADC regular nb conv verification

IS_ADC_REGULAR_NB_CONV

ADC rank into regular group

ADC_REGULAR_RANK_1
ADC_REGULAR_RANK_2
ADC_REGULAR_RANK_3
ADC_REGULAR_RANK_4
ADC_REGULAR_RANK_5
ADC_REGULAR_RANK_6
ADC_REGULAR_RANK_7
ADC_REGULAR_RANK_8
ADC_REGULAR_RANK_9
ADC_REGULAR_RANK_10
ADC_REGULAR_RANK_11
ADC_REGULAR_RANK_12
ADC_REGULAR_RANK_13
ADC_REGULAR_RANK_14
ADC_REGULAR_RANK_15
ADC_REGULAR_RANK_16

ADC sampling times

ADC_SAMPLETIME_1CYCLE_5	Sampling time 1.5 ADC clock cycle
ADC_SAMPLETIME_7CYCLES_5	Sampling time 7.5 ADC clock cycles
ADC_SAMPLETIME_13CYCLES_5	Sampling time 13.5 ADC clock cycles
ADC_SAMPLETIME_28CYCLES_5	Sampling time 28.5 ADC clock cycles

ADC_SAMPLETIME_41CYCLES_5	Sampling time 41.5 ADC clock cycles
ADC_SAMPLETIME_55CYCLES_5	Sampling time 55.5 ADC clock cycles
ADC_SAMPLETIME_71CYCLES_5	Sampling time 71.5 ADC clock cycles
ADC_SAMPLETIME_239CYCLES_5	Sampling time 239.5 ADC clock cycles

ADC sampling times all channels

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT0

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT0

ADC_SAMPLETIME_1CYCLE5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_7CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_13CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_28CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_41CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_55CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_71CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_239CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_1CYCLE5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_7CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_13CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_28CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_41CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_55CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_71CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_239CYCLES5_SMPR1ALLCHANNELS

ADC scan mode

ADC_SCAN_DISABLE

ADC_SCAN_ENABLE

7 HAL ADC Extension Driver

7.1 ADCEx Firmware driver registers structures

7.1.1 ADC_InjectionConfTypeDef

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedNbrOfConversion***
- ***uint32_t InjectedDiscontinuousConvMode***
- ***uint32_t AutoInjectedConv***
- ***uint32_t ExternalTrigInjecConv***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Selection of ADC channel to configure This parameter can be a value of ***ADC_channels*** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Note: On STM32F1 devices with several ADC: Only ADC1 can access internal measurement channels (VrefInt/TempSensor) Note: On STM32F10xx8 and STM32F10xxB devices: A low-amplitude voltage glitch may be generated (on ADC input 0) on the PA0 pin, when the ADC is converting with injection trigger. It is advised to distribute the analog channels so that Channel 0 is configured as an injected channel. Refer to errata sheet of these devices for more details.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Rank in the injected group sequencer This parameter must be a value of ***ADCEx_injected_rank*** Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits). This parameter can be a value of ***ADC_sampling_times*** Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_temp (values rough order: 5us to 17.1us min).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
  Defines the offset to be subtracted from the raw converted data (for channels set on injected group only). Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
  Specifies the number of ranks that will be converted within the injected group

sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
  Specifies whether the conversions sequence of injected group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE. Note: For injected group, number of discontinuous ranks increment is fixed to one-by-one. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv***
  Enables or disables the selected ADC automatic injected group conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of ***ADCEx_External_trigger_source_Injected*** Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behaviour in case of another parameter update on the fly) Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

7.1.2 ADC_MultiModeTypeDef

Data Fields

- ***uint32_t Mode***

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*** Note: In dual mode, a change of channel configuration generates a restart that can produce a loss of synchronization. It is recommended to disable dual mode before any configuration change. Note: In case of simultaneous mode used: Exactly the same sampling time should be configured for

the 2 channels that will be sampled simultaneously by ACD1 and ADC2. Note: In case of interleaved mode used: To avoid overlap between conversions, maximum sampling time allowed is 7 ADC clock cycles for fast interleaved mode and 14 ADC clock cycles for slow interleaved mode. Note: Some multimode parameters are fixed on STM32F1 and can be configured on other STM32 devices with several ADC (multimode configuration structure can have additional parameters). The equivalences are:Parameter 'DMAAccessMode': On STM32F1, this parameter is fixed to 1 DMA channel (one DMA channel for both ADC, DMA of ADC master). On other STM32 devices with several ADC, this is equivalent to parameter 'ADC_DMAACCESSMODE_12_10_BITS'.Parameter 'TwoSamplingDelay': On STM32F1, this parameter is fixed to 7 or 14 ADC clock cycles depending on fast or slow interleaved mode selected. On other STM32 devices with several ADC, this is equivalent to parameter 'ADC_TWOSAMPLINGDELAY_7CYCLES' (for fast interleaved mode).

7.2 ADCEx Firmware driver API description

7.2.1 IO operation functions

This section provides functions allowing to:

- Start conversion of injected group.
- Stop conversion of injected group.
- Poll for conversion complete on injected group.
- Get result of injected channel conversion.
- Start conversion of injected group and enable interruptions.
- Stop conversion of injected group and disable interruptions.
- Start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.
- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.
- Set calibration factors for single or differential ending.

This section contains the following APIs:

- ***HAL_ADCEx_Calibration_Start()***
- ***HAL_ADCEx_InjectedStart()***
- ***HAL_ADCEx_InjectedStop()***
- ***HAL_ADCEx_InjectedPollForConversion()***
- ***HAL_ADCEx_InjectedStart_IT()***
- ***HAL_ADCEx_InjectedStop_IT()***
- ***HAL_ADCEx_MultiModeStart_DMA()***
- ***HAL_ADCEx_MultiModeStop_DMA()***
- ***HAL_ADCEx_InjectedGetValue()***
- ***HAL_ADCEx_MultiModeGetValue()***
- ***HAL_ADCEx_InjectedConvCpltCallback()***

7.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode

This section contains the following APIs:

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

7.2.3 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name **HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc)**

Function description Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status

HAL_ADCEx_InjectedStart

Function name **HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)**

Function description Enables ADC, starts conversion of injected group.

Parameters
- **hadc:** ADC handle

Return values
- **HAL:** status

HAL_ADCEx_InjectedStop

Function name **HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)**

Function description Stop conversion of injected channels.

Parameters
- **hadc:** ADC handle

Return values
- **None:**

Notes
- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- In case of auto-injection mode, HAL_ADC_Stop must be used.

HAL_ADCEx_InjectedPollForConversion

Function name **HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)**

Function description Wait for injected group conversion to be completed.

Parameters
- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values
- **HAL:** status

HAL_ADCEx_InjectedStart_IT

Function name **HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)**

Function description Enables ADC, starts conversion of injected group with interruption.

HAL_ADCEx_InjectedStop_IT

Function name **HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)**

Function description Stop conversion of injected channels, disable interruption of end-of-conversion.

Parameters
- **hadc:** ADC handle

Return values
- **None:**

Notes
- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.

HAL_ADCEx_MultiModeStart_DMA

Function name **HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)**

Function description Enables ADC, starts conversion of regular group and transfers result through DMA.

HAL_ADCEx_MultiModeStop_DMA

Function name **HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)**

Function description Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters
- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values
- **None:**

Notes
- Multimode is kept enabled after this function. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel(), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_ReInit().
- In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_InjectedGetValue

Function name **uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef**

*** hadc, uint32_t InjectedRank)**

Function description: Get ADC injected group conversion result.

Parameters:
- **hadc:** ADC handle
- **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values:
  - ADC_INJECTED_RANK_1: Injected Channel1 selected
  - ADC_INJECTED_RANK_2: Injected Channel2 selected
  - ADC_INJECTED_RANK_3: Injected Channel3 selected
  - ADC_INJECTED_RANK_4: Injected Channel4 selected

Return values:
- **ADC:** group injected conversion data

Notes:
- Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion).
- This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC.If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32 families). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_MultiModeGetValue

Function name: **uint32_t HAL_ADCEx_MultiModeGetValue (ADC_HandleTypeDef * hadc)**

Function description: Returns the last ADC Master&Slave regular conversions results data in the selected multi mode.

Parameters:
- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values:
- **The:** converted data value.

HAL_ADCEx_InjectedConvCpltCallback

Function name: **void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)**

Function description: Injected conversion complete callback in non blocking mode.

Parameters:
- **hadc:** ADC handle

Return values:
- **None:**

HAL_ADCEx_InjectedConfigChannel

Function name: **HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef ***

sConfigInjected)

Function description: Configures the ADC injected group and the selected channel to be linked to the injected group.

Parameters:
- **hadc:** ADC handle
- **sConfigInjected:** Structure of ADC injected group and ADC channel for injected group.

Return values:
- **None:**

Notes:
- Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without reseting the ADC. The setting of these parameters is conditioned to ADC state: this function must be called when ADC is not under conversion.

HAL_ADCEx_MultiModeConfigChannel

Function name: **HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * multimode)**

Function description: Enable ADC multimode and configure multimode parameters.

Parameters:
- **hadc:** ADC handle
- **multimode:** Structure of ADC multimode configuration

Return values:
- **HAL:** status

Notes:
- Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without reseting the ADCs (both ADCs of the common group). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".
- To change back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

7.3 ADCEx Firmware driver defines

7.3.1 ADCEx

ADC Extended Dual ADC Mode

ADC_MODE_INDEPENDENT	ADC dual mode disabled (ADC independent mode)
ADC_DUALMODE_REGSIMULT_INJECSIMULT	ADC dual mode enabled: Combined regular simultaneous + injected simultaneous mode, on groups regular and injected
ADC_DUALMODE_REGSIMULT_ALTERTRIG	ADC dual mode enabled: Combined regular simultaneous + alternate trigger

	mode, on groups regular and injected
ADC_DUALMODE_INJECSIMULT_INTERLFAST	ADC dual mode enabled: Combined injected simultaneous + fast interleaved mode, on groups regular and injected (delay between ADC sampling phases: 7 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INJECSIMULT_INTERLSLOW	ADC dual mode enabled: Combined injected simultaneous + slow Interleaved mode, on groups regular and injected (delay between ADC sampling phases: 14 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INJECSIMULT	ADC dual mode enabled: Injected simultaneous mode, on group injected
ADC_DUALMODE_REGSIMULT	ADC dual mode enabled: Regular simultaneous mode, on group regular
ADC_DUALMODE_INTERLFAST	ADC dual mode enabled: Fast interleaved mode, on group regular (delay between ADC sampling phases: 7 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INTERLSLOW	ADC dual mode enabled: Slow interleaved mode, on group regular (delay between ADC sampling phases: 14 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_ALTERTRIG	ADC dual mode enabled: Alternate trigger mode, on group injected

ADCEx external trigger enable for injected group

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

ADCEx External trigger selection for injected group

ADC_EXTERNALTRIGINJECCONV_T2_TRGO	< List of external triggers with generic trigger name, independently of
ADC_EXTERNALTRIGINJECCONV_T2_CC1	
ADC_EXTERNALTRIGINJECCONV_T3_CC4	
ADC_EXTERNALTRIGINJECCONV_T4_TRGO	
ADC_EXTERNALTRIGINJECCONV_EXT_IT15	External triggers of injected group for

ADC3 only

ADC_EXTERNALTRIGINJECCONV_T4_CC3

ADC_EXTERNALTRIGINJECCONV_T8_CC2

ADC_EXTERNALTRIGINJECCONV_T5_TRGO

ADC_EXTERNALTRIGINJECCONV_T5_CC4

ADC_EXTERNALTRIGINJECCONV_T1_CC4 < External triggers of injected group for all ADC instances

ADC_EXTERNALTRIGINJECCONV_T1_TRGO Note: TIM8_CC4 is available on ADC1 and ADC2 only in high-density and

ADC_EXTERNALTRIGINJECCONV_T8_CC4

ADC_INJECTED_SOFTWARE_START

ADCEx injected nb conv verification

IS_ADC_INJECTED_NB_CONV

ADCEx rank into injected group

ADC_INJECTED_RANK_1

ADC_INJECTED_RANK_2

ADC_INJECTED_RANK_3

ADC_INJECTED_RANK_4

ADC Extended Internal HAL driver trigger selection for injected group

ADC1_2_EXTERNALTRIGINJEC_T2_TRGO

ADC1_2_EXTERNALTRIGINJEC_T2_CC1

ADC1_2_EXTERNALTRIGINJEC_T3_CC4

ADC1_2_EXTERNALTRIGINJEC_T4_TRGO

ADC1_2_EXTERNALTRIGINJEC_EXT_IT15

ADC1_2_EXTERNALTRIGINJEC_T8_CC4

ADC3_EXTERNALTRIGINJEC_T4_CC3

ADC3_EXTERNALTRIGINJEC_T8_CC2

ADC3_EXTERNALTRIGINJEC_T8_CC4

ADC3_EXTERNALTRIGINJEC_T5_TRGO

ADC3_EXTERNALTRIGINJEC_T5_CC4

ADC1_2_3_EXTERNALTRIGINJEC_T1_TRGO

ADC1_2_3_EXTERNALTRIGINJEC_T1_CC4

ADC1_2_3_JSWSTART

ADC Extended Internal HAL driver trigger selection for regular group

ADC1_2_EXTERNALTRIG_T1_CC1

ADC1_2_EXTERNALTRIG_T1_CC2

ADC1_2_EXTERNALTRIG_T2_CC2

ADC1_2_EXTERNALTRIG_T3_TRGO

ADC1_2_EXTERNALTRIG_T4_CC4

ADC1_2_EXTERNALTRIG_EXT_IT11

ADC1_2_EXTERNALTRIG_T8_TRGO

ADC3_EXTERNALTRIG_T3_CC1

ADC3_EXTERNALTRIG_T2_CC3

ADC3_EXTERNALTRIG_T8_CC1

ADC3_EXTERNALTRIG_T8_TRGO

ADC3_EXTERNALTRIG_T5_CC1

ADC3_EXTERNALTRIG_T5_CC3

ADC1_2_3_EXTERNALTRIG_T1_CC3

ADC1_2_3_SWSTART

8 HAL CAN Generic Driver

8.1 CAN Firmware driver registers structures

8.1.1 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

8.1.2 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

8.1.3 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

8.1.4 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

8.2 CAN Firmware driver API description

8.2.1 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 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. Or transmit the desired CAN frame using HAL_CAN_Transmit_IT() function.
6. Receive a CAN frame using HAL_CAN_Receive() function.
7. Or receive a CAN frame using HAL_CAN_Receive_IT() 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

8.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_CAN_Init()***
- ***HAL_CAN_ConfigFilter()***

- ***HAL_CAN_DeInit()***
- ***HAL_CAN_MspInit()***
- ***HAL_CAN_MspDeInit()***

8.2.3 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()***

8.2.4 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()***

8.2.5 Detailed description of functions

HAL_CAN_Init

Function name	**HAL_StatusTypeDef HAL_CAN_Init (CAN_HandleTypeDef * hcan)**
Function description	Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status

HAL_CAN_ConfigFilter

Function name	**HAL_StatusTypeDef HAL_CAN_ConfigFilter (CAN_HandleTypeDef * hcan, CAN_FilterConfTypeDef * sFilterConfig)**

Function description	Configures the CAN reception filter according to the specified parameters in the CAN_FilterInitStruct.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **sFilterConfig:** pointer to a CAN_FilterConfTypeDef structure that contains the filter configuration information.
Return values	• **None:**

HAL_CAN_DeInit

Function name	**HAL_StatusTypeDef HAL_CAN_DeInit (CAN_HandleTypeDef * hcan)**
Function description	Deinitializes the CANx peripheral registers to their default reset values.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status

HAL_CAN_MspInit

Function name	**void HAL_CAN_MspInit (CAN_HandleTypeDef * hcan)**
Function description	Initializes the CAN MSP.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_MspDeInit

Function name	**void HAL_CAN_MspDeInit (CAN_HandleTypeDef * hcan)**
Function description	DeInitializes the CAN MSP.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **None:**

HAL_CAN_Transmit

Function name	**HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * hcan, uint32_t Timeout)**
Function description	Initiates and transmits a CAN frame message.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CAN_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_CAN_Transmit_IT (CAN_HandleTypeDef * hcan)**
Function description	Initiates and transmits a CAN frame message.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status

HAL_CAN_Receive

Function name	**HAL_StatusTypeDef HAL_CAN_Receive (CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)**
Function description	Receives a correct CAN frame.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber:** FIFO Number value • **Timeout:** Specify Timeout value
Return values	• **HAL:** status

HAL_CAN_Receive_IT

Function name	**HAL_StatusTypeDef HAL_CAN_Receive_IT (CAN_HandleTypeDef * hcan, uint8_t FIFONumber)**
Function description	Receives a correct CAN frame.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber:** Specify the FIFO number
Return values	• **HAL:** status

HAL_CAN_Sleep

Function name	**HAL_StatusTypeDef HAL_CAN_Sleep (CAN_HandleTypeDef * hcan)**
Function description	Enters the Sleep (low power) mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL:** status.

HAL_CAN_WakeUp

Function name	**HAL_StatusTypeDef HAL_CAN_WakeUp (CAN_HandleTypeDef * hcan)**
Function description	Wakes up the CAN peripheral from sleep mode, after that the CAN peripheral is in the normal mode.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values
- **HAL:** status.

HAL_CAN_IRQHandler

Function name **void HAL_CAN_IRQHandler (CAN_HandleTypeDef * hcan)**

Function description Handles CAN interrupt request.

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

Return values
- **None:**

HAL_CAN_TxCpltCallback

Function name **void HAL_CAN_TxCpltCallback (CAN_HandleTypeDef * hcan)**

Function description Transmission complete callback in non blocking mode.

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

Return values
- **None:**

HAL_CAN_RxCpltCallback

Function name **void HAL_CAN_RxCpltCallback (CAN_HandleTypeDef * hcan)**

Function description Transmission complete callback in non blocking mode.

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

Return values
- **None:**

HAL_CAN_ErrorCallback

Function name **void HAL_CAN_ErrorCallback (CAN_HandleTypeDef * hcan)**

Function description Error CAN callback.

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

Return values
- **None:**

HAL_CAN_GetError

Function name **uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)**

Function description Return the CAN error code.

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

Return values
- **CAN:** Error Code

HAL_CAN_GetState

Function name **HAL_CAN_StateTypeDef HAL_CAN_GetState**

(CAN_HandleTypeDef * hcan)

Function description: return the CAN state

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

Return values:
- **HAL:** state

8.3 CAN Firmware driver defines

8.3.1 CAN

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_CAN_ERROR_FOV0	FIFO0 overrun error
HAL_CAN_ERROR_FOV1	FIFO1 overrun error

CAN Exported Macros

__HAL_CAN_RESET_HANDLE_STATE

Description:
- Reset CAN handle state.

Parameters:
- __HANDLE__: CAN handle.

Return value:
- None

__HAL_CAN_ENABLE_IT

Description:
- Enable the specified CAN interrupts.

Parameters:
- __HANDLE__: CAN handle.
- __INTERRUPT__: CAN Interrupt. This parameter can be one of the following values:
  - CAN_IT_TME: Transmit mailbox empty interrupt enable
  - 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

Return value:

- None.

__HAL_CAN_DISABLE_IT

Description:

- Disable the specified CAN interrupts.

Parameters:

- __HANDLE__: CAN handle.
- __INTERRUPT__: CAN Interrupt. This parameter can be one of the following values:
  - CAN_IT_TME: Transmit mailbox empty interrupt enable
  - 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

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

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

Return value:

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

__HAL_CAN_CLEAR_FLAG

Description:

- Clear the specified CAN pending flag.

Parameters:

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

Return value:

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

__HAL_CAN_GET_IT_SOURCE

Description:

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

Parameters:

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

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

Return value:

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

__HAL_CAN_FIFO_RELEASE

Description:

- Release the specified receive FIFO.

Parameters:

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

Return value:

- None.

__HAL_CAN_CANCEL_TRANSMIT

Description:

- Cancel a transmit request.

Parameters:

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

Return value:

- None.

__HAL_CAN_DBG_FREEZE

Description:

- Enable or disables the DBG Freeze for CAN.

Parameters:

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

Return value:

- None

CAN Filter FIFO

CAN_FILTER_FIFO0 Filter FIFO 0 assignment for filter x

CAN_FILTER_FIFO1 Filter FIFO 1 assignment for filter x

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 initialization Status

CAN_INITSTATUS_FAILED	CAN initialization failed
CAN_INITSTATUS_SUCCESS	CAN initialization OK

CAN Interrupts

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

CAN_IT_LEC	Last error code interrupt
CAN_IT_ERR	Error Interrupt

CAN 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 Receive FIFO Number

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 Synchronization Jump Width

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

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

CAN Transmit Constants

CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox

9 HAL CAN Extension Driver

9.1 CANEx Firmware driver registers structures

9.1.1 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 = 13.
- ***uint32_t CAN_FilterConfTypeDef::FilterMode***
  Specifies the filter mode to be initialized. This parameter can be a value of ***CAN_filter_mode***
- ***uint32_t CAN_FilterConfTypeDef::FilterScale***
  Specifies the filter scale. This parameter can be a value of ***CAN_filter_scale***
- ***uint32_t CAN_FilterConfTypeDef::FilterActivation***
  Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_FilterConfTypeDef::BankNumber***
  Select the start slave bank filter This parameter must be a number between Min_Data = 0 and Max_Data = 28.

10 HAL CEC Generic Driver

10.1 CEC Firmware driver registers structures

10.1.1 CEC_InitTypeDef

Data Fields

- ***uint32_t TimingErrorFree***
- ***uint32_t PeriodErrorFree***
- ***uint16_t OwnAddress***
- ***uint8_t * RxBuffer***

Field Documentation

- ***uint32_t CEC_InitTypeDef::TimingErrorFree***
  Configures the CEC Bit Timing Error Mode. This parameter can be a value of ***CEC_BitTimingErrorMode***
- ***uint32_t CEC_InitTypeDef::PeriodErrorFree***
  Configures the CEC Bit Period Error Mode. This parameter can be a value of ***CEC_BitPeriodErrorMode***
- ***uint16_t CEC_InitTypeDef::OwnAddress***
  Own addresses configuration This parameter can be a value of ***CEC_OWN_ADDRESS***
- ***uint8_t* CEC_InitTypeDef::RxBuffer***
  CEC Rx buffer pointeur

10.1.2 CEC_HandleTypeDef

Data Fields

- ***CEC_TypeDef * Instance***
- ***CEC_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferCount***
- ***uint16_t RxXferSize***
- ***HAL_LockTypeDef Lock***
- ***HAL_CEC_StateTypeDef gState***
- ***HAL_CEC_StateTypeDef RxState***
- ***uint32_t ErrorCode***

Field Documentation

- ***CEC_TypeDef* CEC_HandleTypeDef::Instance***
  CEC registers base address
- ***CEC_InitTypeDef CEC_HandleTypeDef::Init***
  CEC communication parameters
- ***uint8_t* CEC_HandleTypeDef::pTxBuffPtr***
  Pointer to CEC Tx transfer Buffer
- ***uint16_t CEC_HandleTypeDef::TxXferCount***
  CEC Tx Transfer Counter
- ***uint16_t CEC_HandleTypeDef::RxXferSize***
  CEC Rx Transfer size, 0: header received only
- ***HAL_LockTypeDef CEC_HandleTypeDef::Lock***
  Locking object

- ***HAL_CEC_StateTypeDef CEC_HandleTypeDef::gState***
  CEC state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_CEC_StateTypeDef**
- ***HAL_CEC_StateTypeDef CEC_HandleTypeDef::RxState***
  CEC state information related to Rx operations. This parameter can be a value of **HAL_CEC_StateTypeDef**
- ***uint32_t CEC_HandleTypeDef::ErrorCode***
  For errors handling purposes, copy of ISR register in case error is reported

10.2 CEC Firmware driver API description

10.2.1 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 Bit Timing Error Mode and the Bit Period Error 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 customed HAL_CEC_MspInit() API.

10.2.2 Initialization and Configuration functions

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

- The following parameters need to be configured:
  – TimingErrorFree
  – PeriodErrorFree
  – InitiatorAddress

This section contains the following APIs:

- ***HAL_CEC_Init()***
- ***HAL_CEC_DeInit()***
- ***HAL_CEC_SetDeviceAddress()***
- ***HAL_CEC_MspInit()***
- ***HAL_CEC_MspDeInit()***

10.2.3 IO operation functions

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

1. The CEC handle must contain the initiator (TX side) and the destination (RX side) logical addresses (4-bit long addresses, 0xF for broadcast messages destination)
2. The communication is performed using Interrupts. These API's return the HAL status. The end of the data processing will be indicated through the dedicated CEC IRQ when using Interrupt mode. The HAL_CEC_TxCpltCallback(), HAL_CEC_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_CEC_ErrorCallback() user callback will be executed when a communication error is detected
3. API's with Interrupt are : (+) HAL_CEC_Transmit_IT() (+) HAL_CEC_IRQHandler()
4. A set of User Callbacks are provided: (+) HAL_CEC_TxCpltCallback() (+) HAL_CEC_RxCpltCallback() (+) HAL_CEC_ErrorCallback()

This subsection provides a set of functions allowing to manage the CEC data transfers. (#) The CEC handle must contain the initiator (TX side) and the destination (RX side) logical addresses (4-bit long addresses, 0xF for broadcast messages destination) (#) The communication is performed using Interrupts. These API's return the HAL status. The end of the data processing will be indicated through the dedicated CEC IRQ when using Interrupt mode. The HAL_CEC_TxCpltCallback(), HAL_CEC_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_CEC_ErrorCallback() user callback will be executed when a communication error is detected (#) API's with Interrupt are :

- HAL_CEC_Transmit_IT()
- HAL_CEC_IRQHandler() (#) A set of User Callbacks are provided:
- HAL_CEC_TxCpltCallback()
- HAL_CEC_RxCpltCallback()
- HAL_CEC_ErrorCallback()

This section contains the following APIs:

- ***HAL_CEC_Transmit_IT()***
- ***HAL_CEC_GetLastReceivedFrameSize()***
- ***HAL_CEC_ChangeRxBuffer()***
- ***HAL_CEC_IRQHandler()***
- ***HAL_CEC_TxCpltCallback()***
- ***HAL_CEC_RxCpltCallback()***
- ***HAL_CEC_ErrorCallback()***

10.2.4 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.
- HAL_CEC_GetError() API can be helpful to check in run-time the error of the CEC peripheral.

This section contains the following APIs:

- ***HAL_CEC_GetState()***
- ***HAL_CEC_GetError()***

10.2.5 Detailed description of functions

HAL_CEC_Init

Function name **HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)**

Function description Initializes the CEC mode according to the specified parameters in the CEC_InitTypeDef and creates the associated handle .

Parameters
- **hcec:** CEC handle

Return values
- **HAL:** status

HAL_CEC_DeInit

Function name **HAL_StatusTypeDef HAL_CEC_DeInit (CEC_HandleTypeDef * hcec)**

Function description DeInitializes the CEC peripheral.

Parameters
- **hcec:** CEC handle

Return values
- **HAL:** status

HAL_CEC_SetDeviceAddress

Function name **HAL_StatusTypeDef HAL_CEC_SetDeviceAddress (CEC_HandleTypeDef * hcec, uint16_t CEC_OwnAddress)**

Function description Initializes the Own Address of the CEC device.

Parameters
- **hcec:** CEC handle
- **CEC_OwnAddress:** The CEC own address.

Return values
- **HAL:** status

HAL_CEC_MspInit

Function name **void HAL_CEC_MspInit (CEC_HandleTypeDef * hcec)**

Function description CEC MSP Init.

Parameters
- **hcec:** CEC handle

Return values
- **None:**

HAL_CEC_MspDeInit

Function name **void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)**

Function description CEC MSP DeInit.

Parameters
- **hcec:** CEC handle

Return values
- **None:**

HAL_CEC_Transmit_IT

Function name **HAL_StatusTypeDef HAL_CEC_Transmit_IT (CEC_HandleTypeDef * hcec, uint8_t InitiatorAddress, uint8_t**

DestinationAddress, uint8_t * pData, uint32_t Size)

Function description: Send data in interrupt mode.

Parameters:
- **hcec:** CEC handle
- **InitiatorAddress:** Initiator address
- **DestinationAddress:** destination logical address
- **pData:** pointer to input byte data buffer
- **Size:** amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).

Return values:
- **HAL:** status

HAL_CEC_GetLastReceivedFrameSize

Function name: **uint32_t HAL_CEC_GetLastReceivedFrameSize (CEC_HandleTypeDef * hcec)**

Function description: Get size of the received frame.

Parameters:
- **hcec:** CEC handle

Return values:
- **Frame:** size

HAL_CEC_ChangeRxBuffer

Function name: **void HAL_CEC_ChangeRxBuffer (CEC_HandleTypeDef * hcec, uint8_t * Rxbuffer)**

Function description: Change Rx Buffer.

Parameters:
- **hcec:** CEC handle
- **Rxbuffer:** Rx Buffer

Return values:
- **Frame:** size

Notes:
- This function can be called only inside the HAL_CEC_RxCpltCallback()

HAL_CEC_IRQHandler

Function name: **void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)**

Function description: This function handles CEC interrupt requests.

Parameters:
- **hcec:** CEC handle

Return values:
- **None:**

HAL_CEC_TxCpltCallback

Function name: **void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)**

Function description: Tx Transfer completed callback.

Parameters:
- **hcec:** CEC handle

Return values:
- **None:**

HAL_CEC_RxCpltCallback

Function name	**void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec, uint32_t RxFrameSize)**
Function description	Rx Transfer completed callback.
Parameters	• **hcec:** CEC handle • **RxFrameSize:** Size of frame
Return values	• **None:**

HAL_CEC_ErrorCallback

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

HAL_CEC_GetState

Function name	**HAL_CEC_StateTypeDef HAL_CEC_GetState (CEC_HandleTypeDef * hcec)**
Function description	return the CEC state
Parameters	• **hcec:** pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC module.
Return values	• **HAL:** state

HAL_CEC_GetError

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

10.3 CEC Firmware driver defines

10.3.1 CEC

Bit Period Error Mode

CEC_BIT_PERIOD_ERROR_MODE_STANDARD	Bit period error Standard Mode
CEC_BIT_PERIOD_ERROR_MODE_FLEXIBLE	Bit period error Flexible Mode

Bit Timing Error Mode

CEC_BIT_TIMING_ERROR_MODE_STANDARD	Bit timing error Standard Mode
CEC_BIT_TIMING_ERROR_MODE_ERRORFREE	Bit timing error Free Mode

CEC Error Code

HAL_CEC_ERROR_NONE	no error
HAL_CEC_ERROR_BTE	Bit Timing Error
HAL_CEC_ERROR_BPE	Bit Period Error
HAL_CEC_ERROR_RBTFE	Rx Block Transfer Finished Error
HAL_CEC_ERROR_SBE	Start Bit Error
HAL_CEC_ERROR_ACKE	Block Acknowledge Error
HAL_CEC_ERROR_LINE	Line Error
HAL_CEC_ERROR_TBTFE	Tx Block Transfer Finished Error

CEC Exported Macros

__HAL_CEC_RESET_HANDLE_STATE

Description:

- Reset CEC handle gstate & RxState.

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_TERR: Tx Error
  - CEC_FLAG_TBTRF:Tx Block Transfer Finished
  - CEC_FLAG_RERR: Rx Error
  - CEC_FLAG_RBTF: Rx Block Transfer Finished

Return value:

- ITStatus

__HAL_CEC_CLEAR_FLAG

Description:

- Clears the CEC's pending flags.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the flag to clear. This parameter can be

any combination of the following values:
- CEC_CSR_TERR: Tx Error
- CEC_FLAG_TBTRF: Tx Block Transfer Finished
- CEC_CSR_RERR: Rx Error
- CEC_CSR_RBTF: Rx Block Transfer Finished

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:
  - CEC_IT_IE : Interrupt 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:
  - CEC_IT_IE : Interrupt 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:
  - CEC_IT_IE : Interrupt 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

__HAL_CEC_GET_TRANSMISSION_START_FLAG

Description:

- Get Transmission Start flag.

Parameters:

- __HANDLE__: specifies the

	CEC Handle. **Return value:** • FlagStatus
__HAL_CEC_GET_TRANSMISSION_END_FLAG	**Description:** • Get Transmission End flag. **Parameters:** • __HANDLE__: specifies the CEC Handle. **Return value:** • FlagStatus
__HAL_CEC_CLEAR_OAR	**Description:** • Clear OAR register. **Parameters:** • __HANDLE__: specifies the CEC Handle. **Return value:** • none
__HAL_CEC_SET_OAR	**Description:** • Set OAR register. **Parameters:** • __HANDLE__: specifies the CEC Handle. • __ADDRESS__: Own Address value. **Return value:** • none

Flags definition

CEC_FLAG_TSOM

CEC_FLAG_TEOM

CEC_FLAG_TERR

CEC_FLAG_TBTRF

CEC_FLAG_RSOM

CEC_FLAG_REOM

CEC_FLAG_RERR

CEC_FLAG_RBTF

CEC Initiator logical address position in message header

CEC_INITIATOR_LSB_POS

Interrupts definition

CEC_IT_IE

CEC Own Address

CEC_OWN_ADDRESS_NONE

CEC_OWN_ADDRESS_0

CEC_OWN_ADDRESS_1

CEC_OWN_ADDRESS_2

CEC_OWN_ADDRESS_3

CEC_OWN_ADDRESS_4

CEC_OWN_ADDRESS_5

CEC_OWN_ADDRESS_6

CEC_OWN_ADDRESS_7

CEC_OWN_ADDRESS_8

CEC_OWN_ADDRESS_9

CEC_OWN_ADDRESS_10

CEC_OWN_ADDRESS_11

CEC_OWN_ADDRESS_12

CEC_OWN_ADDRESS_13

CEC_OWN_ADDRESS_14

CEC_OWN_ADDRESS_15

11 HAL CORTEX Generic Driver

11.1 CORTEX Firmware driver registers structures

11.1.1 MPU_Region_InitTypeDef

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint8_t Size***
- ***uint8_t SubRegionDisable***
- ***uint8_t TypeExtField***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t IsShareable***
- ***uint8_t IsCacheable***
- ***uint8_t IsBufferable***

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the number of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::Size***
  Specifies the size of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Size***
- ***uint8_t MPU_Region_InitTypeDef::SubRegionDisable***
  Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint8_t MPU_Region_InitTypeDef::TypeExtField***
  Specifies the TEX field level. This parameter can be a value of ***CORTEX_MPU_TEX_Levels***
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***
- ***uint8_t MPU_Region_InitTypeDef::IsCacheable***
  Specifies the cacheable status of the region protected. This parameter can be a value of ***CORTEX_MPU_Access_Cacheable***

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

11.2 CORTEX Firmware driver API description

11.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

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

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function according to the following table.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ().
4. please refer to programming manual for details in how to configure priority. When the NVIC_PRIORITYGROUP_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the sub priority. IRQ priority order (sorted by highest to lowest priority): Lowest preemption priorityLowest sub priorityLowest 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 stm32f1xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

11.2.2 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()***

11.2.3 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_Disable()***
- ***HAL_MPU_Enable()***
- ***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()***

11.2.4 Detailed description of functions

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

- **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

- **None:**

Notes

- When the NVIC_PriorityGroup_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name	**void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)**
Function description	Sets the priority of an interrupt.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xx.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	• **None:**

HAL_NVIC_EnableIRQ

Function name	**void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)**
Function description	Enables a device specific interrupt in the NVIC interrupt controller.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))
Return values	• **None:**
Notes	• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name	**void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)**
Function description	Disables a device specific interrupt in the NVIC interrupt controller.
Parameters	• **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))
Return values	• **None:**

HAL_NVIC_SystemReset

Function name	**void HAL_NVIC_SystemReset (void)**
Function description	Initiates a system reset request to reset the MCU.
Return values	• **None:**

HAL_SYSTICK_Config

Function name	**uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)**

Function description: Initializes the System Timer and its interrupt, and starts the System Tick Timer.

Parameters:
- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values:
- **status:** - 0 Function succeeded.
  - 1 Function failed.

HAL_NVIC_GetPriorityGrouping

Function name: **uint32_t HAL_NVIC_GetPriorityGrouping (void)**

Function description: Gets the priority grouping field from the NVIC Interrupt Controller.

Return values:
- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name: **void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)**

Function description: Gets the priority of an interrupt.

Parameters:
- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.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:
- **None:**

HAL_NVIC_GetPendingIRQ

Function name: **uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)**

Function description: Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).

Parameters:
- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete

STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values
- **status:** - 0 Interrupt status is not pending.
  – 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name **void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)**

Function description Sets Pending bit of an external interrupt.

Parameters
- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values
- **None:**

HAL_NVIC_ClearPendingIRQ

Function name **void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)**

Function description Clears the pending bit of an external interrupt.

Parameters
- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values
- **None:**

HAL_NVIC_GetActive

Function name **uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)**

Function description Gets active interrupt (reads the active register in NVIC and returns the active bit).

Parameters
- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values
- **status:** - 0 Interrupt status is not pending.
  – 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name **void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)**

Function description Configures the SysTick clock source.

Parameters
- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
  – SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.
  – SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return values
- **None:**

HAL_SYSTICK_IRQHandler

Function name **void HAL_SYSTICK_IRQHandler (void)**

Function description This function handles SYSTICK interrupt request.

Return values
- **None:**

HAL_SYSTICK_Callback

Function name **void HAL_SYSTICK_Callback (void)**

Function description SYSTICK callback.

Return values
- **None:**

HAL_MPU_Enable

Function name **void HAL_MPU_Enable (uint32_t MPU_Control)**

Function description Enable the MPU.

Parameters
- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values
- **None:**

HAL_MPU_Disable

Function name **void HAL_MPU_Disable (void)**

Function description Disables the MPU.

Return values
- **None:**

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
- **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values
- **None:**

11.3 CORTEX Firmware driver defines

11.3.1 CORTEX

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 _SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

12 HAL CRC Generic Driver

12.1 CRC Firmware driver registers structures

12.1.1 CRC_HandleTypeDef

Data Fields

- ***CRC_TypeDef * Instance***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
  Register base address
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
  CRC locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
  CRC communication state

12.2 CRC Firmware driver API description

12.2.1 How to use this driver

The CRC HAL driver can be used as follows:

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

12.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
- DeInitialize the CRC peripheral
- Initialize the CRC MSP
- DeInitialize CRC MSP

This section contains the following APIs:

- ***HAL_CRC_Init()***
- ***HAL_CRC_DeInit()***
- ***HAL_CRC_MspInit()***
- ***HAL_CRC_MspDeInit()***

12.2.3 Peripheral Control functions

This section provides functions allowing to:

- Compute the 32-bit CRC value of 32-bit data buffer, using combination of the previous CRC value and the new one.
- Compute the 32-bit CRC value of 32-bit data buffer, independently of the previous CRC value.

This section contains the following APIs:

- ***HAL_CRC_Accumulate()***
- ***HAL_CRC_Calculate()***

12.2.4 Peripheral State functions

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

This section contains the following APIs:

- ***HAL_CRC_GetState()***

12.2.5 Detailed description of functions

HAL_CRC_Init

Function name	**HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)**
Function description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates the associated handle.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• **HAL:** status

HAL_CRC_DeInit

Function name	**HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)**
Function description	DeInitializes the CRC peripheral.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• **HAL:** status

HAL_CRC_MspInit

Function name	**void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)**
Function description	Initializes the CRC MSP.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• **None:**

HAL_CRC_MspDeInit

Function name	**void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)**
Function description	DeInitializes the CRC MSP.

Parameters
- **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC

Return values
- **None:**

HAL_CRC_Accumulate

Function name **uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**

Function description Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.

Parameters
- **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
- **pBuffer:** pointer to the buffer containing the data to be computed
- **BufferLength:** length of the buffer to be computed (defined in word, 4 bytes)

Return values
- **32-bit:** CRC

HAL_CRC_Calculate

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

Function description Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.

Parameters
- **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
- **pBuffer:** Pointer to the buffer containing the data to be computed
- **BufferLength:** Length of the buffer to be computed (defined in word, 4 bytes)

Return values
- **32-bit:** CRC

HAL_CRC_GetState

Function name **HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)**

Function description Returns the CRC state.

Parameters
- **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC

Return values
- **HAL:** state

12.3 CRC Firmware driver defines

12.3.1 CRC

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:

- Resets CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_SET_IDR

Description:

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

Parameters:

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

Return value:

- None

__HAL_CRC_GET_IDR

Description:

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

Parameters:

- __HANDLE__: CRC handle

Return value:

- 8-bit: value of the ID register

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 ***DACEx_trigger_selection*** Note: For STM32F100x high-density value line devices, additional trigger sources are available.
- ***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_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM4, TIM6, TIM7 For STM32F10x connectivity line devices and STM32F100x devices: TIM3 For STM32F10x high-density and XL-density devices: TIM8 For STM32F100x high-density value line devices: TIM15 as replacement of TIM5. (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 connect feature

Each DAC channel can be connected internally. To connect, use sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_ENABLE;

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel1 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

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 correspondance

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()

DMA requests are mapped as following:

1. DAC channel1 : For STM32F100x low-density, medium-density, high-density with DAC DMA remap: mapped on DMA1 channel3 which must be already configured For STM32F100x high-density without DAC DMA remap and other STM32F1 devices: mapped on DMA2 channel3 which must be already configured
2. DAC channel2 : For STM32F100x low-density, medium-density, high-density with DAC DMA remap: mapped on DMA1 channel4 which must be already configured For STM32F100x high-density without DAC DMA remap and other STM32F1 devices: mapped on DMA2 channel4 which must be already configured

13.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA functions

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, 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 middle of data transfer HAL_DACEx_ConvHalfCpltCallbackCh1()or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1 or HAL_DAC_ConvHalfCpltCallbackCh2
- 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() or HAL_DACEx_ErrorCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1 or HAL_DACEx_ErrorCallbackCh2
- For STM32F100x devices with specific feature: DMA underrun. In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1()or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1 or HAL_DACEx_DMAUnderrunCallbackCh2 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 (For STM32F100x devices with specific feature: DMA underrun)
- __HAL_DAC_DISABLE : Disable the DAC peripheral (For STM32F100x devices with specific feature: DMA underrun)
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags (For STM32F100x devices with specific feature: DMA underrun)
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status (For STM32F100x devices with specific feature: DMA underrun)



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

13.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_DAC_Init()***
- ***HAL_DAC_DeInit()***
- ***HAL_DAC_MspInit()***
- ***HAL_DAC_MspDeInit()***

13.2.4 IO operation functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_DAC_Start()***

- ***HAL_DAC_Stop()***
- ***HAL_DAC_Start_DMA()***
- ***HAL_DAC_Stop_DMA()***
- ***HAL_DAC_GetValue()***
- ***HAL_DAC_ConvCpltCallbackCh1()***
- ***HAL_DAC_ConvHalfCpltCallbackCh1()***
- ***HAL_DAC_ErrorCallbackCh1()***
- ***HAL_DAC_SetValue()***

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_ConvCpltCallbackCh1()***
- ***HAL_DAC_ConvHalfCpltCallbackCh1()***
- ***HAL_DAC_ErrorCallbackCh1()***

13.2.7 Detailed description of functions

HAL_DAC_Init

Function name	**HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)**
Function description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **HAL:** status

HAL_DAC_DeInit

Function name	**HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)**
Function description	Deinitializes the DAC peripheral registers to their default reset values.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that

contains the configuration information for the specified DAC.

Return values
- **HAL:** status

HAL_DAC_MspInit

Function name **void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)**

Function description Initializes the DAC MSP.

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

Return values
- **None:**

HAL_DAC_MspDeInit

Function name **void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)**

Function description DeInitializes the DAC MSP.

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

Return values
- **None:**

HAL_DAC_Start

Function name **HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)**

Function description Enables DAC and starts conversion of channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values
- **HAL:** status

HAL_DAC_Stop

Function name **HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)**

Function description Disables DAC and stop conversion of channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values
- **HAL:** status

HAL_DAC_Start_DMA

Function name
HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description
Enables DAC and starts conversion of channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The Source memory Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values
- **HAL:** status

HAL_DAC_Stop_DMA

Function name
HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description
Disables DAC and stop conversion of channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values
- **HAL:** status

HAL_DAC_SetValue

Function name
HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function description
Set the specified data holding register value for DAC channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_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 selected

  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values
- **HAL:** status

HAL_DAC_GetValue

Function name **uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)**

Function description Returns the last data output value of the selected DAC channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values
- **The:** selected DAC channel data output value.

HAL_DAC_ConfigChannel

Function name **HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)**

Function description Configures the selected DAC channel.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values
- **HAL:** status

HAL_DAC_GetState

Function name **HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)**

Function description return the DAC state

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

Return values
- **HAL:** state

HAL_DAC_GetError

Function name **uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)**

Function description Return the DAC error code.

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that

contains the configuration information for the specified DAC.

Return values
- **DAC:** Error Code

HAL_DAC_ConvCpltCallbackCh1

Function name **void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**

Function description Conversion complete callback in non blocking mode for Channel1.

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

Return values
- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name **void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**

Function description Conversion half DMA transfer callback in non blocking mode for Channel1.

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

Return values
- **None:**

HAL_DAC_ErrorCallbackCh1

Function name **void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)**

Function description Error DAC callback for Channel1.

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

Return values
- **None:**

DAC_DMAConvCpltCh1

Function name **void DAC_DMAConvCpltCh1 (DMA_HandleTypeDef * hdma)**

Function description DMA conversion complete callback.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values
- **None:**

DAC_DMAHalfConvCpltCh1

Function name **void DAC_DMAHalfConvCpltCh1 (DMA_HandleTypeDef * hdma)**

Function description DMA half transfer complete callback.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that

contains the configuration information for the specified DMA module.

Return values
- **None:**

DAC_DMAErrorCh1

Function name **void DAC_DMAErrorCh1 (DMA_HandleTypeDef * hdma)**

Function description DMA error callback.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values
- **None:**

13.3 DAC Firmware driver defines

13.3.1 DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC data alignement

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE	No error
HAL_DAC_ERROR_DMAUNDERRUNCH1	DAC channel1 DMA underrun error
HAL_DAC_ERROR_DMAUNDERRUNCH2	DAC channel2 DMA underrun error
HAL_DAC_ERROR_DMA	DMA error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE **Description:**
- Reset DAC handle state.

Parameters:
- __HANDLE__: 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
DAC output buffer	
DAC_OUTPUTBUFFER_ENABLE	
DAC_OUTPUTBUFFER_DISABLE	

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()***

14.1.3 Detailed description of functions

HAL_DACEx_DualGetValue

Function name	**uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)**
Function description	Returns the last data output value of the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• **The:** selected DAC channel data output value.

HAL_DACEx_TriangleWaveGenerate

Function name	**HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)**
Function description	Enables or disables the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
  - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
  - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
  - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
  - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
  - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
  - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
  - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
  - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
  - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
  - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
  - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
  - DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enables or disables the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
  - DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
  - DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel

LFSR bit[2:0] for noise wave generation
- DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
- DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
- DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
- DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
- DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
- DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
- DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
- DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
- DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values
- **HAL:** status

HAL_DACEx_DualSetValue

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

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

Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel2 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel1 to be loaded in the selected data holding register.

Return values
- **HAL:** status

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

HAL_DACEx_ConvCpltCallbackCh2

Function name
void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description
Conversion complete callback in non blocking mode for Channel2.

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

Return values
- **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name **void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function description Conversion half DMA transfer callback in non blocking mode for Channel2.

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

Return values
- **None:**

HAL_DACEx_ErrorCallbackCh2

Function name **void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function description Error DAC callback for Channel2.

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

Return values
- **None:**

DAC_DMAConvCpltCh2

Function name **void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)**

Function description DMA conversion complete callback.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values
- **None:**

DAC_DMAHalfConvCpltCh2

Function name **void DAC_DMAHalfConvCpltCh2 (DMA_HandleTypeDef * hdma)**

Function description DMA half transfer complete callback.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values
- **None:**

DAC_DMAErrorCh2

Function name **void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)**

Function description DMA error callback.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA

module.

Return values
- **None:**

14.2 DACEx Firmware driver defines

14.2.1 DACEx

DACEx lfsrunmask triangleamplitude

DAC_LFSRUNMASK_BIT0	Unmask DAC channel LFSR bit0 for noise wave generation
DAC_LFSRUNMASK_BITS1_0	Unmask DAC channel LFSR bit[1:0] for noise wave generation
DAC_LFSRUNMASK_BITS2_0	Unmask DAC channel LFSR bit[2:0] for noise wave generation
DAC_LFSRUNMASK_BITS3_0	Unmask DAC channel LFSR bit[3:0] for noise wave generation
DAC_LFSRUNMASK_BITS4_0	Unmask DAC channel LFSR bit[4:0] for noise wave generation
DAC_LFSRUNMASK_BITS5_0	Unmask DAC channel LFSR bit[5:0] for noise wave generation
DAC_LFSRUNMASK_BITS6_0	Unmask DAC channel LFSR bit[6:0] for noise wave generation
DAC_LFSRUNMASK_BITS7_0	Unmask DAC channel LFSR bit[7:0] for noise wave generation
DAC_LFSRUNMASK_BITS8_0	Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0	Unmask DAC channel LFSR bit[9:0] for noise wave generation
DAC_LFSRUNMASK_BITS10_0	Unmask DAC channel LFSR bit[10:0] for noise wave generation
DAC_LFSRUNMASK_BITS11_0	Unmask DAC channel LFSR bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023

DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
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_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_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel
DAC_TRIGGER_T8_TRGO	TIM8 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T5_TRGO	TIM5 TRGO selected as external conversion trigger for DAC channel

15 HAL DMA Generic Driver

15.1 DMA Firmware driver registers structures

15.1.1 DMA_InitTypeDef

Data Fields

- ***uint32_t Direction***
- ***uint32_t PeriphInc***
- ***uint32_t MemInc***
- ***uint32_t PeriphDataAlignment***
- ***uint32_t MemDataAlignment***
- ***uint32_t Mode***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc***
  Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of ***DMA_Peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::MemInc***
  Specifies whether the memory address register should be incremented or not. This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
  Specifies the Peripheral data width. This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
  Specifies the Memory data width. This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the operation mode of the DMAy Channelx. This parameter can be a value of ***DMA_mode***
  Note: The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- ***uint32_t DMA_InitTypeDef::Priority***
  Specifies the software priority for the DMAy Channelx. This parameter can be a value of ***DMA_Priority_level***

15.1.2 __DMA_HandleTypeDef

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***HAL_DMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***

- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***__IO uint32_t ErrorCode***
- ***DMA_TypeDef * DmaBaseAddress***
- ***uint32_t ChannelIndex***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA Half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer abort callback
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA Error code
- ***DMA_TypeDef* __DMA_HandleTypeDef::DmaBaseAddress***
  DMA Channel Base Address
- ***uint32_t __DMA_HandleTypeDef::ChannelIndex***
  DMA Channel Index

15.2 DMA Firmware driver API description

15.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM / FLASH memories: no initialization is necessary). Please refer to the Reference manual for connection between peripherals and DMA requests.
2. For a given Channel, program the required configuration through the following parameters: Channel request, Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode using HAL_DMA_Init() function.
3. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
4. Use HAL_DMA_Abort() function to abort the current transfer In Memory-to-Memory transfer mode, Circular mode is not allowed.

Polling mode IO operation

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

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_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).

DMA HAL driver macros list

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

- __HAL_DMA_ENABLE: Enable the specified DMA Channel.
- __HAL_DMA_DISABLE: Disable the specified DMA Channel.
- __HAL_DMA_GET_FLAG: Get the DMA Channel pending flags.
- __HAL_DMA_CLEAR_FLAG: Clear the DMA Channel pending flags.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Channel interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Channel interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Channel interrupt has occurred or not.



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

15.2.2 Initialization and de-initialization functions

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

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual.

This section contains the following APIs:

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

15.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt

- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- ***HAL_DMA_Start()***
- ***HAL_DMA_Start_IT()***
- ***HAL_DMA_Abort()***
- ***HAL_DMA_Abort_IT()***
- ***HAL_DMA_PollForTransfer()***
- ***HAL_DMA_IRQHandler()***
- ***HAL_DMA_RegisterCallback()***
- ***HAL_DMA_UnRegisterCallback()***

15.2.4 Peripheral 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()***

15.2.5 Detailed description of functions

HAL_DMA_Init

Function name	**HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)**
Function description	Initialize the DMA according to the specified parameters in the DMA_InitTypeDef and initialize the associated handle.
Parameters	• **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_DeInit

Function name	**HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)**
Function description	DeInitialize the DMA peripheral.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **HAL:** status

HAL_DMA_Start

Function name	**HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t**

DataLength)

Function description: Start the DMA Transfer.

Parameters:
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values:
- **HAL:** status

HAL_DMA_Start_IT

Function name: **HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)**

Function description: Start the DMA Transfer with interrupt enabled.

Parameters:
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values:
- **HAL:** status

HAL_DMA_Abort

Function name: **HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)**

Function description: Abort the DMA Transfer.

Parameters:
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values:
- **HAL:** status

HAL_DMA_Abort_IT

Function name: **HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)**

Function description: Aborts the DMA Transfer in Interrupt mode.

Parameters:
- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values:
- **HAL:** status

HAL_DMA_PollForTransfer

Function name	**HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)**
Function description	Polling for transfer complete.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CompleteLevel:** Specifies the DMA level complete. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

HAL_DMA_IRQHandler

Function name	**void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)**
Function description	Handles DMA interrupt request.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• **None:**

HAL_DMA_RegisterCallback

Function name	**HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *_hdma) pCallback)**
Function description	Register callbacks.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter. • **pCallback:** pointer to private callbacsk function which has pointer to a DMA_HandleTypeDef structure as parameter.
Return values	• **HAL:** status

HAL_DMA_UnRegisterCallback

Function name	**HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)**
Function description	UnRegister callbacks.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter.

Return values
- **HAL:** status

HAL_DMA_GetState

Function name **HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)**

Function description Return the DMA hande state.

Parameters
- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values
- **HAL:** state

HAL_DMA_GetError

Function name **uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)**

Function description Return the DMA error code.

Parameters
- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values
- **DMA:** Error Code

15.3 DMA Firmware driver defines

15.3.1 DMA

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_NO_XFER	no ongoing transfer
HAL_DMA_ERROR_TIMEOUT	Timeout error
HAL_DMA_ERROR_NOT_SUPPORTED	Not supported mode

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE **Description:**
- Reset DMA handle state.

Parameters:
- __HANDLE__: DMA handle

Return value:
- None

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_ENABLE_IT

Description:

- Enables the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt

mask

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Check whether the specified DMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_GET_COUNTER

Description:

- Return the number of remaining data units in the current DMA Channel transfer.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: number of remaining data units in the current DMA Channel transfer.

DMA flag definitions

DMA_FLAG_GL1

DMA_FLAG_TC1

DMA_FLAG_HT1

DMA_FLAG_TE1

DMA_FLAG_GL2

DMA_FLAG_TC2

DMA_FLAG_HT2

DMA_FLAG_TE2

DMA_FLAG_GL3

DMA_FLAG_TC3

DMA_FLAG_HT3

DMA_FLAG_TE3

DMA_FLAG_GL4
DMA_FLAG_TC4
DMA_FLAG_HT4
DMA_FLAG_TE4
DMA_FLAG_GL5
DMA_FLAG_TC5
DMA_FLAG_HT5
DMA_FLAG_TE5
DMA_FLAG_GL6
DMA_FLAG_TC6
DMA_FLAG_HT6
DMA_FLAG_TE6
DMA_FLAG_GL7
DMA_FLAG_TC7
DMA_FLAG_HT7
DMA_FLAG_TE7

DMA interrupt enable definitions

DMA_IT_TC
DMA_IT_HT
DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE	Memory data alignment: Byte
DMA_MDATAALIGN_HALFWORD	Memory data alignment: HalfWord
DMA_MDATAALIGN_WORD	Memory data alignment: Word

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 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

16 HAL DMA Extension Driver

16.1 DMAEx Firmware driver defines

16.1.1 DMAEx

DMAEx High density and XL density product devices

__HAL_DMA_GET_TC_FLAG_INDEX

Description:

- Returns the current DMA Channel transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer complete flag index.

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- Returns the current DMA Channel half transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified half transfer complete flag index.

__HAL_DMA_GET_TE_FLAG_INDEX

Description:

- Returns the current DMA Channel transfer error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_GI_FLAG_INDEX

Description:

- Return the current DMA Channel Global interrupt flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_FLAG

Description:

- Get the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx: Transfer complete flag
  - DMA_FLAG_HTx: Half transfer complete flag
  - DMA_FLAG_TEx: Transfer error flag Where x can be 1_7 or 1_5 (depending on DMA1 or DMA2) to select the DMA Channel flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clears the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx: Transfer complete flag
  - DMA_FLAG_HTx: Half transfer complete flag
  - DMA_FLAG_TEx: Transfer error flag Where x can be 1_7 or 1_5 (depending on DMA1 or DMA2) to select the DMA Channel flag.

Return value:

- None

17 HAL ETH Generic Driver

17.1 ETH Firmware driver registers structures

17.1.1 ETH_InitTypeDef

Data Fields

- ***uint32_t AutoNegotiation***
- ***uint32_t Speed***
- ***uint32_t DuplexMode***
- ***uint16_t PhyAddress***
- ***uint8_t * MACAddr***
- ***uint32_t RxMode***
- ***uint32_t ChecksumMode***
- ***uint32_t MediaInterface***

Field Documentation

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

17.1.2 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 then 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 then 2048 bytes to be sent. When disabled, the MAC can send up to 16384 bytes. This parameter can be a value of ***ETH_Jabber***
- ***uint32_t ETH_MACInitTypeDef::InterFrameGap***
  Selects the minimum IFG between frames during transmission. This parameter can be a value of ***ETH_Inter_Frame_Gap***
- ***uint32_t ETH_MACInitTypeDef::CarrierSense***
  Selects or not the Carrier Sense. This parameter can be a value of ***ETH_Carrier_Sense***
- ***uint32_t ETH_MACInitTypeDef::ReceiveOwn***
  Selects or not the ReceiveOwn, ReceiveOwn allows the reception of frames when the TX_EN signal is asserted in Half-Duplex mode. This parameter can be a value of ***ETH_Receive_Own***
- ***uint32_t ETH_MACInitTypeDef::LoopbackMode***
  Selects or not the internal MAC MII Loopback mode. This parameter can be a value of ***ETH_Loop_Back_Mode***
- ***uint32_t ETH_MACInitTypeDef::ChecksumOffload***
  Selects or not the IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. This parameter can be a value of ***ETH_Checksum_Offload***
- ***uint32_t ETH_MACInitTypeDef::RetryTransmission***
  Selects or not the MAC attempt retries transmission, based on the settings of BL, when a collision occurs (Half-Duplex mode). This parameter can be a value of ***ETH_Retry_Transmission***

- ***uint32_t ETH_MACInitTypeDef::AutomaticPadCRCStrip***
  Selects or not the Automatic MAC Pad/CRC Stripping. This parameter can be a value of ***ETH_Automatic_Pad_CRC_Strip***
- ***uint32_t ETH_MACInitTypeDef::BackOffLimit***
  Selects the BackOff limit value. This parameter can be a value of ***ETH_Back_Off_Limit***
- ***uint32_t ETH_MACInitTypeDef::DeferralCheck***
  Selects or not the deferral check function (Half-Duplex mode). This parameter can be a value of ***ETH_Deferral_Check***
- ***uint32_t ETH_MACInitTypeDef::ReceiveAll***
  Selects or not all frames reception by the MAC (No filtering). This parameter can be a value of ***ETH_Receive_All***
- ***uint32_t ETH_MACInitTypeDef::SourceAddrFilter***
  Selects the Source Address Filter mode. This parameter can be a value of ***ETH_Source_Addr_Filter***
- ***uint32_t ETH_MACInitTypeDef::PassControlFrames***
  Sets the forwarding mode of the control frames (including unicast and multicast PAUSE frames) This parameter can be a value of ***ETH_Pass_Control_Frames***
- ***uint32_t ETH_MACInitTypeDef::BroadcastFramesReception***
  Selects or not the reception of Broadcast Frames. This parameter can be a value of ***ETH_Broadcast_Frames_Reception***
- ***uint32_t ETH_MACInitTypeDef::DestinationAddrFilter***
  Sets the destination filter mode for both unicast and multicast frames. This parameter can be a value of ***ETH_Destination_Addr_Filter***
- ***uint32_t ETH_MACInitTypeDef::PromiscuousMode***
  Selects or not the Promiscuous Mode This parameter can be a value of ***ETH_Promiscuous_Mode***
- ***uint32_t ETH_MACInitTypeDef::MulticastFramesFilter***
  Selects the Multicast Frames filter mode: None/HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of ***ETH_Multicast_Frames_Filter***
- ***uint32_t ETH_MACInitTypeDef::UnicastFramesFilter***
  Selects the Unicast Frames filter mode: HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of ***ETH_Unicast_Frames_Filter***
- ***uint32_t ETH_MACInitTypeDef::HashTableHigh***
  This field holds the higher 32 bits of Hash table. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFFFFFFU
- ***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 = 0xFFFFFFFFU
- ***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 = 0xFFFFU
- ***uint32_t ETH_MACInitTypeDef::ZeroQuantaPause***
  Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of ***ETH_Zero_Quanta_Pause***
- ***uint32_t ETH_MACInitTypeDef::PauseLowThreshold***
  This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of ***ETH_Pause_Low_Threshold***
- ***uint32_t ETH_MACInitTypeDef::UnicastPauseFrameDetect***
  Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast

address and unique multicast address). This parameter can be a value of ***ETH_Unicast_Pause_Frame_Detect***

- ***uint32_t ETH_MACInitTypeDef::ReceiveFlowControl***
  Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of ***ETH_Receive_Flow_Control***
- ***uint32_t ETH_MACInitTypeDef::TransmitFlowControl***
  Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of ***ETH_Transmit_Flow_Control***
- ***uint32_t ETH_MACInitTypeDef::VLANTagComparison***
  Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of ***ETH_VLAN_Tag_Comparison***
- ***uint32_t ETH_MACInitTypeDef::VLANTagIdentifier***
  Holds the VLAN tag identifier for receive frames

17.1.3 ETH_DMAInitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t ETH_DMAInitTypeDef::DropTCPIPChecksumErrorFrame***
  Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of ***ETH_Drop_TCP_IP_Checksum_Error_Frame***
- ***uint32_t ETH_DMAInitTypeDef::ReceiveStoreForward***
  Enables or disables the Receive store and forward mode. This parameter can be a value of ***ETH_Receive_Store_Forward***
- ***uint32_t ETH_DMAInitTypeDef::FlushReceivedFrame***
  Enables or disables the flushing of received frames. This parameter can be a value of ***ETH_Flush_Received_Frame***
- ***uint32_t ETH_DMAInitTypeDef::TransmitStoreForward***
  Enables or disables Transmit store and forward mode. This parameter can be a value of ***ETH_Transmit_Store_Forward***
- ***uint32_t ETH_DMAInitTypeDef::TransmitThresholdControl***
  Selects or not the Transmit Threshold Control. This parameter can be a value of ***ETH_Transmit_Threshold_Control***
- ***uint32_t ETH_DMAInitTypeDef::ForwardErrorFrames***
  Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of ***ETH_Forward_Error_Frames***

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

17.1.4 ETH_DMADescTypeDef

Data Fields

- ***__IO uint32_t Status***
- ***uint32_t ControlBufferSize***
- ***uint32_t Buffer1Addr***
- ***uint32_t Buffer2NextDescAddr***

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

17.1.5 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

17.1.6 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

17.2 ETH Firmware driver API description

17.2.1 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. The PTP protocol and the DMA descriptors ring mode are not supported in this driver

17.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- ***HAL_ETH_Init()***
- ***HAL_ETH_DeInit()***
- ***HAL_ETH_DMATxDescListInit()***
- ***HAL_ETH_DMARxDescListInit()***
- ***HAL_ETH_MspInit()***
- ***HAL_ETH_MspDeInit()***

17.2.3 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()***

17.2.4 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()***

17.2.5 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()***

17.2.6 Detailed description of functions

HAL_ETH_Init

Function name	**HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)**
Function description	Initializes the Ethernet MAC and DMA according to default parameters.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• **HAL:** status

HAL_ETH_DeInit

Function name	**HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)**
Function description	De-Initializes the ETH peripheral.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values
- **HAL:** status

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:**

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:**

HAL_ETH_DMATxDescListInit

Function name **HAL_StatusTypeDef HAL_ETH_DMATxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMATxDescTab, uint8_t * TxBuff, uint32_t TxBuffCount)**

Function description Initializes the DMA Tx descriptors in chain mode.

Parameters
- **heth:** pointer to a ETH_HandleTypeDef 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

Return values
- **HAL:** status

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

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

HAL_ETH_GetReceivedFrame

Function name: **HAL_StatusTypeDef HAL_ETH_GetReceivedFrame (ETH_HandleTypeDef * heth)**

Function description: Checks for received frames.

Parameters:
- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values:
- **HAL:** status

HAL_ETH_ReadPHYRegister

Function name: **HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t * RegValue)**

Function description: Reads a PHY register.

Parameters:
- **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:
- **HAL:** status

HAL_ETH_WritePHYRegister

Function name: **HAL_StatusTypeDef HAL_ETH_WritePHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t RegValue)**

Function description: Writes to a PHY register.

Parameters:
- **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:
- **HAL:** status

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	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• **HAL:** status

HAL_ETH_IRQHandler

Function name	**void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)**
Function description	This function handles ETH interrupt request.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• **HAL:** status

HAL_ETH_TxCpltCallback

Function name	**void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)**
Function description	Tx Transfer completed callbacks.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• **None:**

HAL_ETH_RxCpltCallback

Function name	**void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)**
Function description	Rx Transfer completed callbacks.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• **None:**

HAL_ETH_ErrorCallback

Function name	**void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)**
Function description	Ethernet transfer error callbacks.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• **None:**

HAL_ETH_Start

Function name	**HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)**

Function description: Enables Ethernet MAC and DMA reception/transmission.

Parameters:
- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values:
- **HAL:** status

HAL_ETH_Stop

Function name: **HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)**

Function description: Stop Ethernet MAC and DMA reception/transmission.

Parameters:
- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values:
- **HAL:** status

HAL_ETH_ConfigMAC

Function name: **HAL_StatusTypeDef HAL_ETH_ConfigMAC (ETH_HandleTypeDef * heth, ETH_MACInitTypeDef * macconf)**

Function description: Set ETH MAC Configuration.

Parameters:
- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** MAC Configuration structure

Return values:
- **HAL:** status

HAL_ETH_ConfigDMA

Function name: **HAL_StatusTypeDef HAL_ETH_ConfigDMA (ETH_HandleTypeDef * heth, ETH_DMAInitTypeDef * dmaconf)**

Function description: Sets ETH DMA Configuration.

Parameters:
- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** DMA Configuration structure

Return values:
- **HAL:** status

HAL_ETH_GetState

Function name: **HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)**

Function description: Return the ETH HAL state.

Parameters:
- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values:
- **HAL:** state

17.3 ETH Firmware driver defines

17.3.1 ETH

ETH Address Aligned Beats

ETH_ADDRESSALIGNEDBEATS_ENABLE

ETH_ADDRESSALIGNEDBEATS_DISABLE

ETH Automatic Pad CRC Strip

ETH_AUTOMATICPADCRCSTRIP_ENABLE

ETH_AUTOMATICPADCRCSTRIP_DISABLE

ETH AutoNegotiation

ETH_AUTONEGOTIATION_ENABLE

ETH_AUTONEGOTIATION_DISABLE

ETH Back Off Limit

ETH_BACKOFFLIMIT_10

ETH_BACKOFFLIMIT_8

ETH_BACKOFFLIMIT_4

ETH_BACKOFFLIMIT_1

ETH Broadcast Frames Reception

ETH_BROADCASTFRAMESRECEPTION_ENABLE

ETH_BROADCASTFRAMESRECEPTION_DISABLE

ETH Buffers setting

ETH_MAX_PACKET_SIZE	ETH_HEADER + ETH_EXTRA + ETH_VLAN_TAG + ETH_MAX_ETH_PAYLOAD + ETH_CRC
ETH_HEADER	6 byte Dest addr, 6 byte Src addr, 2 byte length/type
ETH_CRC	Ethernet CRC
ETH_EXTRA	Extra bytes in some cases
ETH_VLAN_TAG	optional 802.1q VLAN Tag
ETH_MIN_ETH_PAYLOAD	Minimum Ethernet payload size
ETH_MAX_ETH_PAYLOAD	Maximum Ethernet payload size
ETH_JUMBO_FRAME_PAYLOAD	Jumbo frame payload size

ETH_RX_BUF_SIZE

ETH_RXBUFNB

ETH_TX_BUF_SIZE

ETH_TXBUFNB

ETH Carrier Sense

ETH_CARRIERSENCE_ENABLE

ETH_CARRIERSENCE_DISABLE

ETH Checksum Mode

ETH_CHECKSUM_BY_HARDWARE

ETH_CHECKSUM_BY_SOFTWARE

ETH Checksum Offload

ETH_CHECKSUMOFFLAOD_ENABLE

ETH_CHECKSUMOFFLAOD_DISABLE

ETH Deferral Check

ETH_DEFFERRALCHECK_ENABLE

ETH_DEFFERRALCHECK_DISABLE

ETH Destination Addr Filter

ETH_DESTINATIONADDRFILTER_NORMAL

ETH_DESTINATIONADDRFILTER_INVERSE

ETH DMA Arbitration

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_1_1

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_2_1

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_3_1

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_4_1

ETH_DMAARBITRATION_RXPRIORTX

ETH DMA Flags

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

ETH_DMA_FLAG_TJT	Transmit jabber timeout flag
ETH_DMA_FLAG_TBU	Transmit buffer unavailable flag
ETH_DMA_FLAG_TPS	Transmit process stopped flag
ETH_DMA_FLAG_T	Transmit flag

ETH DMA Interrupts

ETH_DMA_IT_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_IT_PMT	PMT interrupt (on DMA)
ETH_DMA_IT_MMC	MMC interrupt (on DMA)
ETH_DMA_IT_NIS	Normal interrupt summary
ETH_DMA_IT_AIS	Abnormal interrupt summary
ETH_DMA_IT_ER	Early receive interrupt
ETH_DMA_IT_FBE	Fatal bus error interrupt
ETH_DMA_IT_ET	Early transmit interrupt
ETH_DMA_IT_RWT	Receive watchdog timeout interrupt
ETH_DMA_IT_RPS	Receive process stopped interrupt
ETH_DMA_IT_RBU	Receive buffer unavailable interrupt
ETH_DMA_IT_R	Receive interrupt
ETH_DMA_IT_TU	Underflow interrupt
ETH_DMA_IT_RO	Overflow interrupt
ETH_DMA_IT_TJT	Transmit jabber timeout interrupt
ETH_DMA_IT_TBU	Transmit buffer unavailable interrupt
ETH_DMA_IT_TPS	Transmit process stopped interrupt
ETH_DMA_IT_T	Transmit interrupt

ETH DMA overflow

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

ETH DMA receive process state

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

ETH DMA RX Descriptor

ETH_DMARXDESC_OWN	OWN bit: descriptor is owned by DMA engine
ETH_DMARXDESC_AFM	DA Filter Fail for the rx frame
ETH_DMARXDESC_FL	Receive descriptor frame length
ETH_DMARXDESC_ES	Error summary: OR of the following bits: DE || OE || IPC || LC || RWT || RE || CE
ETH_DMARXDESC_DE	Descriptor error: no more descriptors for receive frame
ETH_DMARXDESC_SAF	SA Filter Fail for the received frame
ETH_DMARXDESC_LE	Frame size not matching with length field
ETH_DMARXDESC_OE	Overflow Error: Frame was damaged due to buffer overflow
ETH_DMARXDESC_VLAN	VLAN Tag: received frame is a VLAN frame
ETH_DMARXDESC_FS	First descriptor of the frame
ETH_DMARXDESC_LS	Last descriptor of the frame
ETH_DMARXDESC_IPV4HCE	IPC Checksum Error: Rx Ipv4 header checksum error
ETH_DMARXDESC_LC	Late collision occurred during reception
ETH_DMARXDESC_FT	Frame type - Ethernet, otherwise 802.3
ETH_DMARXDESC_RWT	Receive Watchdog Timeout: watchdog timer expired during reception
ETH_DMARXDESC_RE	Receive error: error reported by MII interface
ETH_DMARXDESC_DBE	Dribble bit error: frame contains non int multiple of 8 bits
ETH_DMARXDESC_CE	CRC error
ETH_DMARXDESC_MAMPCE	Rx MAC Address/Payload Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error
ETH_DMARXDESC_DIC	Disable Interrupt on Completion
ETH_DMARXDESC_RBS2	Receive Buffer2 Size
ETH_DMARXDESC_RER	Receive End of Ring
ETH_DMARXDESC_RCH	Second Address Chained
ETH_DMARXDESC_RBS1	Receive Buffer1 Size
ETH_DMARXDESC_B1AP	Buffer1 Address Pointer
ETH_DMARXDESC_B2AP	Buffer2 Address Pointer

ETH 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 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_DROPTCPIPCHECKSUMERRORFRAME_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

__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_ENABLE

Description:

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

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_DISABLE

Description:

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

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_MAC_ENABLE_IT

Description:

- Enables the specified ETHERNET MAC interrupts.

Parameters:

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

Return value:

- None

__HAL_ETH_MAC_DISABLE_IT

Description:

- Disables the specified ETHERNET MAC interrupts.

Parameters:

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

Return value:

- 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_MMC : 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_RXFIFOCOUNTER : Overflow for FIFO Overflows Counter
  - ETH_DMA_OVERFLOW_MISSEDFRAMECOUNTER : 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_WAKEUP_ENABLE

Description:

- Enables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_GLOBAL_UNICAST_WAKEUP_DISABLE

Description:

- Disables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_WAKEUP_FRAME_DETECTION_ENABLE

Description:

- Enables the MAC Wake-Up Frame Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_WAKEUP_FRAME_DETECTION_DISABLE

Description:

- Disables the MAC Wake-Up Frame Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_MAGIC_PACKET_DETECTION_ENABLE

Description:

- Enables the MAC Magic Packet Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_MAGIC_PACKET_DETECTION_DISABLE

Description:

- Disables the MAC Magic Packet Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_POWER_DOWN_ENABLE

Description:

- Enables the MAC Power Down.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_POWER_DOWN_DISABLE

Description:

- Disables the MAC Power Down.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_GET_PMT_FLAG_STATUS

Description:

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

Parameters:

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

Return value:

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

__HAL_ETH_MMC_COUNTER_FULL_PRESET

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_PRESET

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_ONREAD_ENABLE	**Description:** • Enables the MMC Reset On Read. **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None
__HAL_ETH_ETH_MMC_RESET_ONREAD_DISABLE	**Description:** • Disables the MMC Reset On Read. **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None
__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_ENABLE	**Description:** • Enables the MMC Counter Stop Rollover. **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None
__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_DISABLE	**Description:** • Disables the MMC Counter Stop Rollover. **Parameters:** • __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_MMC_COUNTERS_RESET

Description:

- Resets the MMC Counters.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None

__HAL_ETH_MMC_RX_IT_ENABLE

Description:

- Enables the specified ETHERNET MMC Rx interrupts.

Parameters:

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

Return value:

- None

__HAL_ETH_MMC_RX_IT_DISABLE

Description:

- Disables the specified ETHERNET MMC Rx interrupts.

Parameters:

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

maximum value

Return value:

- None

__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_GENERATE_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 [70]

ETH MAC addresses filter SA DA

ETH_MAC_ADDRESSFILTER_SA

ETH_MAC_ADDRESSFILTER_DA

ETH MAC Flags

ETH_MAC_FLAG_TST	Time stamp trigger flag (on MAC)

ETH_MAC_FLAG_MMCT	MMC transmit flag
ETH_MAC_FLAG_MMCR	MMC receive flag
ETH_MAC_FLAG_MMC	MMC flag (on MAC)
ETH_MAC_FLAG_PMT	PMT flag (on MAC)

ETH MAC Interrupts

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

ETH Media Interface

ETH_MEDIA_INTERFACE_MII

ETH_MEDIA_INTERFACE_RMII

ETH MMC Rx Interrupts

ETH_MMC_IT_RGUF	When Rx good unicast frames counter reaches half the maximum value
ETH_MMC_IT_RFAE	When Rx alignment error counter reaches half the maximum value
ETH_MMC_IT_RFCE	When Rx crc error counter reaches half the maximum value

ETH MMC Tx Interrupts

ETH_MMC_IT_TGF	When Tx good frame counter reaches half the maximum value
ETH_MMC_IT_TGFMSC	When Tx good multi col counter reaches half the maximum value
ETH_MMC_IT_TGFSC	When Tx good single col counter reaches half the maximum value

ETH Multicast Frames Filter

ETH_MULTICASTFRAMESFILTER_PERFECTHASHTABLE

ETH_MULTICASTFRAMESFILTER_HASHTABLE

ETH_MULTICASTFRAMESFILTER_PERFECT

ETH_MULTICASTFRAMESFILTER_NONE

ETH Pass Control Frames

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

control frames that pass the Address Filter.

ETH Pause Low Threshold

ETH_PAUSELOWTHRESHOLD_MINUS4	Pause time minus 4 slot times
ETH_PAUSELOWTHRESHOLD_MINUS28	Pause time minus 28 slot times
ETH_PAUSELOWTHRESHOLD_MINUS144	Pause time minus 144 slot times
ETH_PAUSELOWTHRESHOLD_MINUS256	Pause time minus 256 slot times

ETH PMT Flags

ETH_PMT_FLAG_WUFFRPR	Wake-Up Frame Filter Register Pointer Reset
ETH_PMT_FLAG_WUFR	Wake-Up Frame Received
ETH_PMT_FLAG_MPR	Magic Packet Received

ETH 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

18 HAL FLASH Generic Driver

18.1 FLASH Firmware driver registers structures

18.1.1 FLASH_ProcessTypeDef

Data Fields

- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t DataRemaining***
- ***__IO uint32_t Address***
- ***__IO uint64_t Data***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
  Internal variable to indicate which procedure is ongoing or not in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::DataRemaining***
  Internal variable to save the remaining pages to erase or half-word to program in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
  Internal variable to save address selected for program or erase
- ***__IO uint64_t FLASH_ProcessTypeDef::Data***
  Internal variable to save data to be programmed
- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
  FLASH locking object
- ***__IO uint32_t FLASH_ProcessTypeDef::ErrorCode***
  FLASH error code This parameter can be a value of ***FLASH_Error_Codes***

18.2 FLASH Firmware driver API description

18.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface accelerates code execution with a system of instruction prefetch.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- Option Bytes programming

18.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F1xx devices.

1. FLASH Memory I/O Programming functions: this group includes all needed functions to erase and program the main memory:
   – Lock and Unlock the FLASH interface
   – Erase function: Erase page, erase all pages
   – Program functions: half word, word and doubleword
2. FLASH Option Bytes Programming functions: this group includes all needed functions to manage the Option Bytes:
   – Lock and Unlock the Option Bytes
   – Set/Reset the write protection
   – Set the Read protection Level
   – Program the user Option Bytes
   – Launch the Option Bytes loader
   – Erase Option Bytes
   – Program the data Option Bytes
   – Get the Write protection.
   – Get the user option bytes.
3. Interrupts and flags management functions : this group includes all needed functions to:
   – Handle FLASH interrupts
   – Wait for last FLASH operation according to its status
   – Get error flag status

In addition to these function, this driver includes a set of macros allowing to handle the following operations:

- Set/Get the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the half cycle access
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

18.2.3 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()***

18.2.4 Peripheral Errors functions

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

This section contains the following APIs:

- ***HAL_FLASH_GetError()***

18.2.5 Detailed description of functions

HAL_FLASH_Program

Function name **HAL_StatusTypeDef HAL_FLASH_Program (uint32_t**

TypeProgram, uint32_t Address, uint64_t Data)

Function description	Program halfword, word or double word at a specified address.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • **Address:** Specifies the address to be programmed. • **Data:** Specifies the data to be programmed
Return values	• **HAL_StatusTypeDef:** HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface • If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one. • FLASH should be previously erased before new programmation (only exception to this is when 0x0000 is programmed)

HAL_FLASH_Program_IT

Function name	**HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)**
Function description	Program halfword, word or double word at a specified address with interrupt enabled.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • **Address:** Specifies the address to be programmed. • **Data:** Specifies the data to be programmed
Return values	• **HAL_StatusTypeDef:** HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface • If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.

HAL_FLASH_IRQHandler

Function name	**void HAL_FLASH_IRQHandler (void)**
Function description	This function handles FLASH interrupt request.
Return values	• **None:**

HAL_FLASH_EndOfOperationCallback

Function name	**void HAL_FLASH_EndOfOperationCallback (uint32_t**

ReturnValue)

Function description: FLASH end of operation interrupt callback.

Parameters:
- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure
  - Mass Erase: No return value expected
  - Pages Erase: Address of the page which has been erased (if 0xFFFFFFFF, it means that all the selected pages have been erased)
  - Program: Address which was selected for data program

Return values:
- **none:**

HAL_FLASH_OperationErrorCallback

Function name: **void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)**

Function description: FLASH operation error interrupt callback.

Parameters:
- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure
  - Mass Erase: No return value expected
  - Pages Erase: Address of the page which returned an error
  - Program: Address which was selected for data program

Return values:
- **none:**

HAL_FLASH_Unlock

Function name: **HAL_StatusTypeDef HAL_FLASH_Unlock (void)**

Function description: Unlock the FLASH control register access.

Return values:
- **HAL:** Status

HAL_FLASH_Lock

Function name: **HAL_StatusTypeDef HAL_FLASH_Lock (void)**

Function description: Locks the FLASH control register access.

Return values:
- **HAL:** Status

HAL_FLASH_OB_Unlock

Function name: **HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)**

Function description: Unlock the FLASH Option Control Registers access.

Return values:
- **HAL:** Status

HAL_FLASH_OB_Lock

Function name: **HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)**

Function description: Lock the FLASH Option Control Registers access.

Return values
- **HAL:** Status

HAL_FLASH_OB_Launch

Function name **HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)**

Function description Launch the option byte loading.

Return values
- **HAL:** Status

Notes
- This function will reset automatically the MCU.

HAL_FLASH_GetError

Function name **uint32_t HAL_FLASH_GetError (void)**

Function description Get the specific FLASH error flag.

Return values
- **FLASH_ErrorCode:** The returned value can be: FLASH Error Codes

FLASH_WaitForLastOperation

Function name **HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout)**

Function description Wait for a FLASH operation to complete.

Parameters
- **Timeout:** maximum flash operation timeout

Return values
- **HAL:** Status

FLASH_WaitForLastOperationBank2

Function name **HAL_StatusTypeDef FLASH_WaitForLastOperationBank2 (uint32_t Timeout)**

Function description Wait for a FLASH BANK2 operation to complete.

Parameters
- **Timeout:** maximum flash operation timeout

Return values
- **HAL_StatusTypeDef:** HAL Status

18.3 FLASH Firmware driver defines

18.3.1 FLASH

FLASH Error Codes

HAL_FLASH_ERROR_NONE	No error
HAL_FLASH_ERROR_PROG	Programming error
HAL_FLASH_ERROR_WRP	Write protection error
HAL_FLASH_ERROR_OPTV	Option validity error

Flag definition

FLASH_FLAG_BSY	FLASH Bank1 Busy flag
FLASH_FLAG_PGERR	FLASH Bank1 Programming error flag

FLASH_FLAG_WRPERR	FLASH Bank1 Write protected error flag
FLASH_FLAG_EOP	FLASH Bank1 End of Operation flag
FLASH_FLAG_BSY_BANK1	FLASH Bank1 Busy flag
FLASH_FLAG_PGERR_BANK1	FLASH Bank1 Programming error flag
FLASH_FLAG_WRPERR_BANK1	FLASH Bank1 Write protected error flag
FLASH_FLAG_EOP_BANK1	FLASH Bank1 End of Operation flag
FLASH_FLAG_BSY_BANK2	FLASH Bank2 Busy flag
FLASH_FLAG_PGERR_BANK2	FLASH Bank2 Programming error flag
FLASH_FLAG_WRPERR_BANK2	FLASH Bank2 Write protected error flag
FLASH_FLAG_EOP_BANK2	FLASH Bank2 End of Operation flag
FLASH_FLAG_OPTVERR	Option Byte Error

FLASH Half Cycle

__HAL_FLASH_HALF_CYCLE_ACCESS_ENABLE

Description:

- Enable the FLASH half cycle access.

Return value:

- None

Notes:

- half cycle access can only be used with a low-frequency clock of less than 8 MHz that can be obtained with the use of HSI or HSE but not of PLL.

__HAL_FLASH_HALF_CYCLE_ACCESS_DISABLE

Description:

- Disable the FLASH half cycle access.

Return value:

- None

Notes:

- half cycle access can only be used with a low-frequency clock of less than 8 MHz that can be obtained with the use of HSI or HSE but not of PLL.

Interrupt

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

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

  - FLASH_IT_EOP_BANK1 End of FLASH Operation Interrupt on bank1
  - FLASH_IT_ERR_BANK1 Error Interrupt on bank1
  - FLASH_IT_EOP_BANK2 End of FLASH Operation Interrupt on bank2
  - FLASH_IT_ERR_BANK2 Error Interrupt on bank2

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_BANK1 End of FLASH Operation Interrupt on bank1
  - FLASH_IT_ERR_BANK1 Error Interrupt on bank1
  - FLASH_IT_EOP_BANK2 End of FLASH Operation Interrupt on bank2
  - FLASH_IT_ERR_BANK2 Error Interrupt on bank2

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_BANK1 FLASH End of Operation flag on bank1
  - FLASH_FLAG_WRPERR_BANK1 FLASH Write protected error flag on bank1
  - FLASH_FLAG_PGERR_BANK1 FLASH Programming error flag on bank1
  - FLASH_FLAG_BSY_BANK1 FLASH Busy flag on bank1
  - FLASH_FLAG_EOP_BANK2 FLASH End of Operation flag on bank2
  - FLASH_FLAG_WRPERR_BANK2 FLASH Write protected error flag on bank2
  - FLASH_FLAG_PGERR_BANK2 FLASH Programming error flag on bank2
  - FLASH_FLAG_BSY_BANK2 FLASH Busy flag on bank2
  - FLASH_FLAG_OPTVERR Loaded OB and its

complement do not match

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_BANK1 FLASH End of Operation flag on bank1
  - FLASH_FLAG_WRPERR_BANK1 FLASH Write protected error flag on bank1
  - FLASH_FLAG_PGERR_BANK1 FLASH Programming error flag on bank1
  - FLASH_FLAG_BSY_BANK1 FLASH Busy flag on bank1
  - FLASH_FLAG_EOP_BANK2 FLASH End of Operation flag on bank2
  - FLASH_FLAG_WRPERR_BANK2 FLASH Write protected error flag on bank2
  - FLASH_FLAG_PGERR_BANK2 FLASH Programming error flag on bank2
  - FLASH_FLAG_BSY_BANK2 FLASH Busy flag on bank2
  - FLASH_FLAG_OPTVERR Loaded OB and its complement do not match

Return value:

- none

Interrupt definition

FLASH_IT_EOP	End of FLASH Operation Interrupt source Bank1
FLASH_IT_ERR	Error Interrupt source Bank1
FLASH_IT_EOP_BANK1	End of FLASH Operation Interrupt source Bank1
FLASH_IT_ERR_BANK1	Error Interrupt source Bank1
FLASH_IT_EOP_BANK2	End of FLASH Operation Interrupt source Bank2
FLASH_IT_ERR_BANK2	Error Interrupt source Bank2

FLASH Latency

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

FLASH Prefetch

__HAL_FLASH_PREFETCH_BUFFER_ENABLE **Description:**

- Enable the FLASH prefetch

	buffer. **Return value:** • None
__HAL_FLASH_PREFETCH_BUFFER_DISABLE	**Description:** • Disable the FLASH prefetch buffer. **Return value:** • None

FLASH Type Program

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

19 HAL FLASH Extension Driver

19.1 FLASHEx Firmware driver registers structures

19.1.1 FLASH_EraseInitTypeDef

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t PageAddress***
- ***uint32_t NbPages***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***
  TypeErase: Mass erase or page erase. This parameter can be a value of ***FLASHEx_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks***
  Select banks to erase when Mass erase is enabled. This parameter must be a value of ***FLASHEx_Banks***
- ***uint32_t FLASH_EraseInitTypeDef::PageAddress***
  PageAdress: Initial FLASH page address to erase when mass erase is disabled This parameter must be a number between Min_Data = 0x08000000 and Max_Data = FLASH_BANKx_END (x = 1 or 2 depending on devices)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  NbPages: Number of pagess to be erased. This parameter must be a value between Min_Data = 1 and Max_Data = (max number of pages - value of initial page)

19.1.2 FLASH_OBProgramInitTypeDef

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPState***
- ***uint32_t WRPPage***
- ***uint32_t Banks***
- ***uint8_t RDPLevel***
- ***uint8_t USERConfig***
- ***uint32_t DATAAddress***
- ***uint8_t DATAData***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***
  OptionType: Option byte to be configured. This parameter can be a value of ***FLASHEx_OB_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState***
  WRPState: Write protection activation or deactivation. This parameter can be a value of ***FLASHEx_OB_WRP_State***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPPage***
  WRPPage: specifies the page(s) to be write protected This parameter can be a value of ***FLASHEx_OB_Write_Protection***

- ***uint32_t FLASH_OBProgramInitTypeDef::Banks***
  Select banks for WRP activation/deactivation of all sectors. This parameter must be a value of ***FLASHEx_Banks***
- ***uint8_t FLASH_OBProgramInitTypeDef::RDPLevel***
  RDPLevel: Set the read protection level.. This parameter can be a value of ***FLASHEx_OB_Read_Protection***
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig***
  USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 This parameter can be a combination of ***FLASHEx_OB_IWatchdog***, ***FLASHEx_OB_nRST_STOP***, ***FLASHEx_OB_nRST_STDBY***, ***FLASHEx_OB_BOOT1***
- ***uint32_t FLASH_OBProgramInitTypeDef::DATAAddress***
  DATAAddress: Address of the option byte DATA to be programmed This parameter can be a value of ***FLASHEx_OB_Data_Address***
- ***uint8_t FLASH_OBProgramInitTypeDef::DATAData***
  DATAData: Data to be stored in the option byte DATA This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF

19.2 FLASHEx Firmware driver API description

19.2.1 FLASH Erasing Programming functions

The FLASH Memory Erasing functions, includes the following functions:

- @ref HAL_FLASHEx_Erase: return only when erase has been done
- @ref HAL_FLASHEx_Erase_IT: end of erase is done when @ref HAL_FLASH_EndOfOperationCallback is called with parameter 0xFFFFFFFF

Any operation of erase should follow these steps:

1. Call the @ref HAL_FLASH_Unlock() function to enable the flash control register and program memory access.
2. Call the desired function to erase page.
3. Call the @ref HAL_FLASH_Lock() to disable the flash program memory access (recommended to protect the FLASH memory against possible unwanted operation).

This section contains the following APIs:

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

19.2.2 Option Bytes Programming functions

This subsection provides a set of functions allowing to control the FLASH option bytes operations.

This section contains the following APIs:

- ***HAL_FLASHEx_OBErase()***
- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***
- ***HAL_FLASHEx_OBGetUserData()***

19.2.3 Detailed description of functions

HAL_FLASHEx_Erase

Function name **HAL_StatusTypeDef HAL_FLASHEx_Erase**

(FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)

Function description: Perform a mass erase or erase the specified FLASH memory pages.

Parameters:
- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **PageError:** pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased)

Return values:
- **HAL_StatusTypeDef:** HAL Status

Notes:
- To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

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 pages with interrupt enabled.

Parameters:
- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.

Return values:
- **HAL_StatusTypeDef:** HAL Status

Notes:
- To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

HAL_FLASHEx_OBErase

Function name: **HAL_StatusTypeDef HAL_FLASHEx_OBErase (void)**

Function description: Erases the FLASH option bytes.

Return values:
- **HAL:** status

Notes:
- This functions erases all option bytes except the Read protection (RDP). The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)

HAL_FLASHEx_OBProgram

Function name: **HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)**

Function description: Program option bytes.

Parameters:
- **pOBInit:** pointer to an FLASH_OBInitStruct structure that

contains the configuration information for the programming.

Return values
- **HAL_StatusTypeDef:** HAL Status

Notes
- The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)

HAL_FLASHEx_OBGetConfig

Function name **void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)**

Function description Get the Option byte configuration.

Parameters
- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.

Return values
- **None:**

HAL_FLASHEx_OBGetUserData

Function name **uint32_t HAL_FLASHEx_OBGetUserData (uint32_t DATAAdress)**

Function description Get the Option byte user data.

Parameters
- **DATAAdress:** Address of the option byte DATA This parameter can be one of the following values:
  – OB_DATA_ADDRESS_DATA0
  – OB_DATA_ADDRESS_DATA1

Return values
- **Value:** programmed in USER data

19.3 FLASHEx Firmware driver defines

19.3.1 FLASHEx

Banks

FLASH_BANK_1 Bank 1

FLASH_BANK_2 Bank 2

FLASH_BANK_BOTH Bank1 and Bank2

Option Byte BOOT1

OB_BOOT1_RESET BOOT1 Reset

OB_BOOT1_SET BOOT1 Set

Option Byte Data Address

OB_DATA_ADDRESS_DATA0

OB_DATA_ADDRESS_DATA1

Option Byte IWatchdog

OB_IWDG_SW	Software IWDG selected
OB_IWDG_HW	Hardware IWDG selected

Option Byte nRST STDBY

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

Option Byte nRST STOP

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

Option Byte Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

Option Bytes Type

OPTIONBYTE_WRP	WRP option byte configuration
OPTIONBYTE_RDP	RDP option byte configuration
OPTIONBYTE_USER	USER option byte configuration
OPTIONBYTE_DATA	DATA option byte configuration

Option Bytes Write Protection

OB_WRP_PAGES0TO1	Write protection of page 0 TO 1
OB_WRP_PAGES2TO3	Write protection of page 2 TO 3
OB_WRP_PAGES4TO5	Write protection of page 4 TO 5
OB_WRP_PAGES6TO7	Write protection of page 6 TO 7
OB_WRP_PAGES8TO9	Write protection of page 8 TO 9
OB_WRP_PAGES10TO11	Write protection of page 10 TO 11
OB_WRP_PAGES12TO13	Write protection of page 12 TO 13
OB_WRP_PAGES14TO15	Write protection of page 14 TO 15
OB_WRP_PAGES16TO17	Write protection of page 16 TO 17
OB_WRP_PAGES18TO19	Write protection of page 18 TO 19
OB_WRP_PAGES20TO21	Write protection of page 20 TO 21
OB_WRP_PAGES22TO23	Write protection of page 22 TO 23
OB_WRP_PAGES24TO25	Write protection of page 24 TO 25
OB_WRP_PAGES26TO27	Write protection of page 26 TO 27
OB_WRP_PAGES28TO29	Write protection of page 28 TO 29
OB_WRP_PAGES30TO31	Write protection of page 30 TO 31
OB_WRP_PAGES32TO33	Write protection of page 32 TO 33
OB_WRP_PAGES34TO35	Write protection of page 34 TO 35
OB_WRP_PAGES36TO37	Write protection of page 36 TO 37

OB_WRP_PAGES38TO39	Write protection of page 38 TO 39
OB_WRP_PAGES40TO41	Write protection of page 40 TO 41
OB_WRP_PAGES42TO43	Write protection of page 42 TO 43
OB_WRP_PAGES44TO45	Write protection of page 44 TO 45
OB_WRP_PAGES46TO47	Write protection of page 46 TO 47
OB_WRP_PAGES48TO49	Write protection of page 48 TO 49
OB_WRP_PAGES50TO51	Write protection of page 50 TO 51
OB_WRP_PAGES52TO53	Write protection of page 52 TO 53
OB_WRP_PAGES54TO55	Write protection of page 54 TO 55
OB_WRP_PAGES56TO57	Write protection of page 56 TO 57
OB_WRP_PAGES58TO59	Write protection of page 58 TO 59
OB_WRP_PAGES60TO61	Write protection of page 60 TO 61
OB_WRP_PAGES62TO127	Write protection of page 62 TO 127
OB_WRP_PAGES62TO255	Write protection of page 62 TO 255
OB_WRP_PAGES62TO511	Write protection of page 62 TO 511
OB_WRP_ALLPAGES	Write protection of all Pages
OB_WRP_PAGES0TO15MASK	
OB_WRP_PAGES16TO31MASK	
OB_WRP_PAGES32TO47MASK	
OB_WRP_PAGES48TO511MASK	

Option Byte WRP State

OB_WRPSTATE_DISABLE	Disable the write protection of the desired pages
OB_WRPSTATE_ENABLE	Enable the write protection of the desired pagess

Page Size

FLASH_PAGE_SIZE

Type Erase

FLASH_TYPEERASE_PAGES	Pages erase only
FLASH_TYPEERASE_MASSERASE	Flash mass erase activation

20 HAL GPIO Generic Driver

20.1 GPIO Firmware driver registers structures

20.1.1 GPIO_InitTypeDef

Data Fields

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

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***

20.2 GPIO Firmware driver API description

20.2.1 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 20 edge detectors in connectivity line devices, or 19 edge detectors in other devices for generating event/interrupt requests. Each input line can be independently configured to select the type (event or interrupt) and the corresponding trigger event (rising or falling or both). Each line can also masked independently. A pending register maintains the status line of the interrupt requests

20.2.2 How to use this driver

1. Enable the GPIO APB2 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
   - 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 PD0 and PD1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

20.2.3 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()***

20.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the GPIOs.

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()***

20.2.5 Detailed description of functions

HAL_GPIO_Init

Function name	**void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)**
Function description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• **None:**

HAL_GPIO_DeInit

Function name	**void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)**
Function description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) 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).
Return values	• **None:**

HAL_GPIO_ReadPin

Function name	**GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Reads the specified input port pin.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	• **The:** input port pin value.

HAL_GPIO_WritePin

Function name	**void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**
Function description	Sets or clears the selected data port bit.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) 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_BIT_RESET: to clear the port pin
  - GPIO_BIT_SET: to set the port pin

Return values
- **None:**

Notes
- This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_TogglePin

Function name **void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**

Function description Toggles the specified GPIO pin.

Parameters
- **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral
- **GPIO_Pin:** Specifies the pins to be toggled.

Return values
- **None:**

HAL_GPIO_LockPin

Function name **HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**

Function description Locks GPIO Pins configuration registers.

Parameters
- **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral
- **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values
- **None:**

Notes
- The locking mechanism allows the IO configuration to be frozen. When the LOCK sequence has been applied on a port bit, it is no longer possible to modify the value of the port bit until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name **void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)**

Function description This function handles EXTI interrupt request.

Parameters
- **GPIO_Pin:** Specifies the pins connected EXTI line

Return values
- **None:**

HAL_GPIO_EXTI_Callback

Function name **void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)**

Function description EXTI line detection callbacks.

Parameters
- **GPIO_Pin:** Specifies the pins connected EXTI line

Return values
- **None:**

20.3 GPIO Firmware driver defines

20.3.1 GPIO

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:
- Checks whether the specified EXTI line flag is set or not.

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

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

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:
- Clears the EXTI's line pending flags.

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

Return value:
- None

__HAL_GPIO_EXTI_GET_IT

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

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

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

__HAL_GPIO_EXTI_CLEAR_IT

Description:
- Clears the EXTI's line pending bits.

Parameters:
- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any

	combination of GPIO_PIN_x where x can be (0..15) **Return value:** • None
__HAL_GPIO_EXTI_GENERATE_SWIT	**Description:** • Generates a Software interrupt on selected EXTI line. **Parameters:** • __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15) **Return value:** • None

GPIO mode define

GPIO_MODE_INPUT	Input Floating Mode
GPIO_MODE_OUTPUT_PP	Output Push Pull Mode
GPIO_MODE_OUTPUT_OD	Output Open Drain Mode
GPIO_MODE_AF_PP	Alternate Function Push Pull Mode
GPIO_MODE_AF_OD	Alternate Function Open Drain Mode
GPIO_MODE_AF_INPUT	Alternate Function Input 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 pull define

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

GPIO speed define

GPIO_SPEED_FREQ_LOW	Low speed
GPIO_SPEED_FREQ_MEDIUM	Medium speed
GPIO_SPEED_FREQ_HIGH	High speed

21 HAL GPIO Extension Driver

21.1 GPIOEx Firmware driver API description

21.1.1 GPIO Peripheral extension features

GPIO module on STM32F1 family, manage also the AFIO register:

- Possibility to use the EVENTOUT Cortex feature

21.1.2 How to use this driver

This driver provides functions to use EVENTOUT Cortex feature

1. Configure EVENTOUT Cortex feature using the function HAL_GPIOEx_ConfigEventout()
2. Activate EVENTOUT Cortex feature using the HAL_GPIOEx_EnableEventout()
3. Deactivate EVENTOUT Cortex feature using the HAL_GPIOEx_DisableEventout()

21.1.3 Extended features functions

This section provides functions allowing to:

- Configure EVENTOUT Cortex feature using the function HAL_GPIOEx_ConfigEventout()
- Activate EVENTOUT Cortex feature using the HAL_GPIOEx_EnableEventout()
- Deactivate EVENTOUT Cortex feature using the HAL_GPIOEx_DisableEventout()

This section contains the following APIs:

- ***HAL_GPIOEx_ConfigEventout()***
- ***HAL_GPIOEx_EnableEventout()***
- ***HAL_GPIOEx_DisableEventout()***

21.1.4 Detailed description of functions

HAL_GPIOEx_ConfigEventout

Function name	**void HAL_GPIOEx_ConfigEventout (uint32_t GPIO_PortSource, uint32_t GPIO_PinSource)**
Function description	Configures the port and pin on which the EVENTOUT Cortex signal will be connected.
Parameters	• **GPIO_PortSource:** Select the port used to output the Cortex EVENTOUT signal. This parameter can be a value of EVENTOUT Port. • **GPIO_PinSource:** Select the pin used to output the Cortex EVENTOUT signal. This parameter can be a value of EVENTOUT Pin.
Return values	• **None:**

HAL_GPIOEx_EnableEventout

Function name	**void HAL_GPIOEx_EnableEventout (void)**

Function description	Enables the Event Output.
Return values	• **None:**

HAL_GPIOEx_DisableEventout

Function name	**void HAL_GPIOEx_DisableEventout (void)**
Function description	Disables the Event Output.
Return values	• **None:**

21.2 GPIOEx Firmware driver defines

21.2.1 GPIOEx

Alternate Function Remapping

__HAL_AFIO_REMAP_SPI1_ENABLE

Description:

- Enable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

Return value:

- None

Notes:

- ENABLE: Remap (NSS/PA15, SCK/PB3, MISO/PB4, MOSI/PB5)

__HAL_AFIO_REMAP_SPI1_DISABLE

Description:

- Disable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

Return value:

- None

Notes:

- DISABLE: No remap (NSS/PA4, SCK/PA5, MISO/PA6, MOSI/PA7)

__HAL_AFIO_REMAP_I2C1_ENABLE

Description:

- Enable the remapping of I2C1 alternate function SCL and SDA.

Return value:

- None

Notes:

- ENABLE: Remap (SCL/PB8, SDA/PB9)

__HAL_AFIO_REMAP_I2C1_DISABLE

Description:

- Disable the remapping of I2C1 alternate function SCL and SDA.

Return value:

- None

Notes:

- DISABLE: No remap (SCL/PB6, SDA/PB7)

__HAL_AFIO_REMAP_USART1_ENABLE

Description:

- Enable the remapping of USART1 alternate function TX and RX.

Return value:

- None

Notes:

- ENABLE: Remap (TX/PB6, RX/PB7)

__HAL_AFIO_REMAP_USART1_DISABLE

Description:

- Disable the remapping of USART1 alternate function TX and RX.

Return value:

- None

Notes:

- DISABLE: No remap (TX/PA9, RX/PA10)

__HAL_AFIO_REMAP_USART2_ENABLE

Description:

- Enable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None

Notes:

- ENABLE: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)

__HAL_AFIO_REMAP_USART2_DISABLE

Description:

- Disable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None

Notes:

- DISABLE: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)

__HAL_AFIO_REMAP_USART3_ENABLE

Description:

- Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None

Notes:

- ENABLE: Full remap (TX/PD8, RX/PD9, CK/PD10, CTS/PD11, RTS/PD12)

__HAL_AFIO_REMAP_USART3_PARTIAL

Description:

- Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None

Notes:

- PARTIAL: Partial remap (TX/PC10, RX/PC11, CK/PC12, CTS/PB13, RTS/PB14)

__HAL_AFIO_REMAP_USART3_DISABLE

Description:

- Disable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None

Notes:

- DISABLE: No remap (TX/PB10, RX/PB11, CK/PB12, CTS/PB13, RTS/PB14)

__HAL_AFIO_REMAP_TIM1_ENABLE

Description:

- Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger

(ETR) and Break input (BKIN)

Return value:

- None

Notes:

- ENABLE: Full remap (ETR/PE7, CH1/PE9, CH2/PE11, CH3/PE13, CH4/PE14, BKIN/PE15, CH1N/PE8, CH2N/PE10, CH3N/PE12)

__HAL_AFIO_REMAP_TIM1_PARTIAL

Description:

- Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

Return value:

- None

Notes:

- PARTIAL: Partial remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PA6, CH1N/PA7, CH2N/PB0, CH3N/PB1)

__HAL_AFIO_REMAP_TIM1_DISABLE

Description:

- Disable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

Return value:

- None

Notes:

- DISABLE: No remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PB12, CH1N/PB13, CH2N/PB14, CH3N/PB15)

__HAL_AFIO_REMAP_TIM2_ENABLE

Description:

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None

Notes:

- ENABLE: Full remap (CH1/ETR/PA15, CH2/PB3, CH3/PB10, CH4/PB11)

__HAL_AFIO_REMAP_TIM2_PARTIAL_2

Description:

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None

Notes:

- PARTIAL_2: Partial remap (CH1/ETR/PA0, CH2/PA1, CH3/PB10, CH4/PB11)

__HAL_AFIO_REMAP_TIM2_PARTIAL_1

Description:

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None

Notes:

- PARTIAL_1: Partial remap (CH1/ETR/PA15, CH2/PB3, CH3/PA2, CH4/PA3)

__HAL_AFIO_REMAP_TIM2_DISABLE

Description:

- Disable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None

Notes:

- DISABLE: No remap (CH1/ETR/PA0, CH2/PA1, CH3/PA2, CH4/PA3)

__HAL_AFIO_REMAP_TIM3_ENABLE

Description:

- Enable the remapping of TIM3 alternate function channels 1 to 4.

Return value:

- None

Notes:

- ENABLE: Full remap (CH1/PC6, CH2/PC7, CH3/PC8, CH4/PC9) TIM3_ETR on PE0 is not re-mapped.

__HAL_AFIO_REMAP_TIM3_PARTIAL

Description:

- Enable the remapping of TIM3 alternate function channels 1 to 4.

Return value:

- None

Notes:

- PARTIAL: Partial remap (CH1/PB4, CH2/PB5, CH3/PB0, CH4/PB1) TIM3_ETR on PE0 is not re-mapped.

__HAL_AFIO_REMAP_TIM3_DISABLE

Description:

- Disable the remapping of TIM3 alternate function channels 1 to 4.

Return value:

- None

Notes:

- DISABLE: No remap (CH1/PA6, CH2/PA7, CH3/PB0, CH4/PB1) TIM3_ETR on PE0 is not re-mapped.

__HAL_AFIO_REMAP_TIM4_ENABLE

Description:

- Enable the remapping of TIM4 alternate function channels 1 to 4.

Return value:

- None

Notes:

- ENABLE: Full remap (TIM4_CH1/PD12, TIM4_CH2/PD13, TIM4_CH3/PD14, TIM4_CH4/PD15) TIM4_ETR on PE0 is not re-mapped.

__HAL_AFIO_REMAP_TIM4_DISABLE

Description:

- Disable the remapping of TIM4 alternate function channels 1 to 4.

Return value:

- None

Notes:

- DISABLE: No remap (TIM4_CH1/PB6, TIM4_CH2/PB7, TIM4_CH3/PB8, TIM4_CH4/PB9) TIM4_ETR on PE0 is not re-mapped.

__HAL_AFIO_REMAP_CAN1_1

Description:

- Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return value:

- None

Notes:

- CASE 1: CAN_RX mapped to PA11, CAN_TX mapped to PA12

__HAL_AFIO_REMAP_CAN1_2

Description:

- Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return value:

- None

Notes:

- CASE 2: CAN_RX mapped to PB8, CAN_TX mapped to PB9 (not available on 36-pin package)

__HAL_AFIO_REMAP_CAN1_3

Description:

- Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return value:

- None

Notes:

- CASE 3: CAN_RX mapped to PD0, CAN_TX mapped to PD1

__HAL_AFIO_REMAP_PD01_ENABLE

Description:

- Enable the remapping of PD0 and PD1.

Return value:

- None

Notes:

- ENABLE: PD0 remapped on OSC_IN, PD1 remapped on OSC_OUT.

__HAL_AFIO_REMAP_PD01_DISABLE

Description:

- Disable the remapping of PD0 and PD1.

Return value:

- None

Notes:

- DISABLE: No remapping of PD0 and PD1

__HAL_AFIO_REMAP_TIM5CH4_ENABLE

Description:

- Enable the remapping of TIM5CH4.

Return value:

- None

Notes:

- ENABLE: LSI internal clock is connected to TIM5_CH4 input for calibration purpose. This function is available only in high density value line devices.

__HAL_AFIO_REMAP_TIM5CH4_DISABLE

Description:

- Disable the remapping of TIM5CH4.

Return value:

- None

Notes:

- DISABLE: TIM5_CH4 is

connected to PA3 This function is available only in high density value line devices.

__HAL_AFIO_REMAP_ADC1_ETRGINJ_ENABLE

Description:

- Enable the remapping of ADC1_ETRGINJ (ADC 1 External trigger injected conversion).

Return value:

- None

Notes:

- ENABLE: ADC1 External Event injected conversion is connected to TIM8 Channel4.

__HAL_AFIO_REMAP_ADC1_ETRGINJ_DISABLE

Description:

- Disable the remapping of ADC1_ETRGINJ (ADC 1 External trigger injected conversion).

Return value:

- None

Notes:

- DISABLE: ADC1 External trigger injected conversion is connected to EXTI15

__HAL_AFIO_REMAP_ADC1_ETRGREG_ENABLE

Description:

- Enable the remapping of ADC1_ETRGREG (ADC 1 External trigger regular conversion).

Return value:

- None

Notes:

- ENABLE: ADC1 External Event regular conversion is connected to TIM8 TRG0.

__HAL_AFIO_REMAP_ADC1_ETRGREG_DISABLE

Description:

- Disable the remapping of ADC1_ETRGREG (ADC 1 External trigger regular conversion).

Return value:

- None

Notes:

- DISABLE: ADC1 External trigger regular conversion is connected to EXTI11

__HAL_AFIO_REMAP_ADC2_ETRGINJ_ENABLE

Description:

- Enable the remapping of ADC2_ETRGREG (ADC 2 External trigger injected conversion).

Return value:

- None

Notes:

- ENABLE: ADC2 External Event injected conversion is connected to TIM8 Channel4.

__HAL_AFIO_REMAP_ADC2_ETRGINJ_DISABLE

Description:

- Disable the remapping of ADC2_ETRGREG (ADC 2 External trigger injected conversion).

Return value:

- None

Notes:

- DISABLE: ADC2 External trigger injected conversion is connected to EXTI15

__HAL_AFIO_REMAP_ADC2_ETRGREG_ENABLE

Description:

- Enable the remapping of ADC2_ETRGREG (ADC 2 External trigger regular conversion).

Return value:

- None

Notes:

- ENABLE: ADC2 External Event regular conversion is connected to TIM8 TRG0.

__HAL_AFIO_REMAP_ADC2_ETRGREG_DISABLE

Description:

- Disable the remapping of ADC2_ETRGREG (ADC 2 External trigger regular conversion).

Return value:

- None

Notes:

- DISABLE: ADC2 External trigger regular conversion is connected to EXTI11

__HAL_AFIO_REMAP_SWJ_ENABLE

Description:

- Enable the Serial wire JTAG configuration.

Return value:

- None

Notes:

- ENABLE: Full SWJ (JTAG-DP + SW-DP): Reset State

__HAL_AFIO_REMAP_SWJ_NONJTRST

Description:

- Enable the Serial wire JTAG configuration.

Return value:

- None

Notes:

- NONJTRST: Full SWJ (JTAG-DP + SW-DP) but without NJTRST

__HAL_AFIO_REMAP_SWJ_NOJTAG

Description:

- Enable the Serial wire JTAG configuration.

Return value:

- None

Notes:

- NOJTAG: JTAG-DP Disabled and SW-DP Enabled

__HAL_AFIO_REMAP_SWJ_DISABLE

Description:

- Disable the Serial wire JTAG configuration.

Return value:

- None

Notes:

- DISABLE: JTAG-DP Disabled and SW-DP Disabled

__HAL_AFIO_REMAP_TIM9_ENABLE

Description:

- Enable the remapping of TIM9_CH1 and TIM9_CH2.

Return value:

- None

Notes:

- ENABLE: Remap (TIM9_CH1 on PE5 and TIM9_CH2 on PE6).

__HAL_AFIO_REMAP_TIM9_DISABLE

Description:

- Disable the remapping of TIM9_CH1 and TIM9_CH2.

Return value:

- None

Notes:

- DISABLE: No remap (TIM9_CH1 on PA2 and TIM9_CH2 on PA3).

__HAL_AFIO_REMAP_TIM10_ENABLE

Description:

- Enable the remapping of TIM10_CH1.

Return value:

- None

Notes:

- ENABLE: Remap (TIM10_CH1 on PF6).

__HAL_AFIO_REMAP_TIM10_DISABLE

Description:

- Disable the remapping of TIM10_CH1.

Return value:

- None

Notes:

- DISABLE: No remap (TIM10_CH1 on PB8).

__HAL_AFIO_REMAP_TIM11_ENABLE

Description:

- Enable the remapping of TIM11_CH1.

Return value:

- None

Notes:

- ENABLE: Remap

(TIM11_CH1 on PF7).

__HAL_AFIO_REMAP_TIM11_DISABLE

Description:

- Disable the remapping of TIM11_CH1.

Return value:

- None

Notes:

- DISABLE: No remap (TIM11_CH1 on PB9).

__HAL_AFIO_REMAP_TIM13_ENABLE

Description:

- Enable the remapping of TIM13_CH1.

Return value:

- None

Notes:

- ENABLE: Remap STM32F100:(TIM13_CH1 on PF8). Others:(TIM13_CH1 on PB0).

__HAL_AFIO_REMAP_TIM13_DISABLE

Description:

- Disable the remapping of TIM13_CH1.

Return value:

- None

Notes:

- DISABLE: No remap STM32F100:(TIM13_CH1 on PA6). Others:(TIM13_CH1 on PC8).

__HAL_AFIO_REMAP_TIM14_ENABLE

Description:

- Enable the remapping of TIM14_CH1.

Return value:

- None

Notes:

- ENABLE: Remap STM32F100:(TIM14_CH1 on PB1). Others:(TIM14_CH1 on PF9).

__HAL_AFIO_REMAP_TIM14_DISABLE

Description:

- Disable the remapping of

TIM14_CH1.

Return value:

- None

Notes:

- DISABLE: No remap STM32F100:(TIM14_CH1 on PC9). Others:(TIM14_CH1 on PA7).

__HAL_AFIO_FSMCNADV_DISCONNECTED

Description:

- Controls the use of the optional FSMC_NADV signal.

Return value:

- None

Notes:

- DISCONNECTED: The NADV signal is not connected. The I/O pin can be used by another peripheral.

__HAL_AFIO_FSMCNADV_CONNECTED

Description:

- Controls the use of the optional FSMC_NADV signal.

Return value:

- None

Notes:

- CONNECTED: The NADV signal is connected to the output (default).

EVENTOUT Pin

AFIO_EVENTOUT_PIN_0	EVENTOUT on pin 0
AFIO_EVENTOUT_PIN_1	EVENTOUT on pin 1
AFIO_EVENTOUT_PIN_2	EVENTOUT on pin 2
AFIO_EVENTOUT_PIN_3	EVENTOUT on pin 3
AFIO_EVENTOUT_PIN_4	EVENTOUT on pin 4
AFIO_EVENTOUT_PIN_5	EVENTOUT on pin 5
AFIO_EVENTOUT_PIN_6	EVENTOUT on pin 6
AFIO_EVENTOUT_PIN_7	EVENTOUT on pin 7
AFIO_EVENTOUT_PIN_8	EVENTOUT on pin 8
AFIO_EVENTOUT_PIN_9	EVENTOUT on pin 9
AFIO_EVENTOUT_PIN_10	EVENTOUT on pin 10

AFIO_EVENTOUT_PIN_11	EVENTOUT on pin 11
AFIO_EVENTOUT_PIN_12	EVENTOUT on pin 12
AFIO_EVENTOUT_PIN_13	EVENTOUT on pin 13
AFIO_EVENTOUT_PIN_14	EVENTOUT on pin 14
AFIO_EVENTOUT_PIN_15	EVENTOUT on pin 15
IS_AFIO_EVENTOUT_PIN	

EVENTOUT Port

AFIO_EVENTOUT_PORT_A	EVENTOUT on port A
AFIO_EVENTOUT_PORT_B	EVENTOUT on port B
AFIO_EVENTOUT_PORT_C	EVENTOUT on port C
AFIO_EVENTOUT_PORT_D	EVENTOUT on port D
AFIO_EVENTOUT_PORT_E	EVENTOUT on port E
IS_AFIO_EVENTOUT_PORT	

22 HAL HCD Generic Driver

22.1 HCD Firmware driver registers structures

22.1.1 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[15U]***
  Host channels parameters
- ***HAL_LockTypeDef HCD_HandleTypeDef::Lock***
  HCD peripheral status
- ***__IO HCD_StateTypeDef HCD_HandleTypeDef::State***
  HCD communication state
- ***void* HCD_HandleTypeDef::pData***
  Pointer Stack Handler

22.2 HCD Firmware driver API description

22.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macro
      – __HAL_RCC_USB_OTG_FS_CLK_ENABLE()
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

22.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_HCD_Init()***
- ***HAL_HCD_HC_Init()***
- ***HAL_HCD_HC_Halt()***
- ***HAL_HCD_DeInit()***
- ***HAL_HCD_MspInit()***
- ***HAL_HCD_MspDeInit()***

22.2.3 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()***

22.2.4 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()***

22.2.5 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()***

22.2.6 Detailed description of functions

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

HAL_HCD_DeInit

Function name	**HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)**

Function description	DeInitialize the host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_HC_Init

Function name	**HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)**
Function description	Initialize a host channel.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15 • **epnum:** 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_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	• **HAL:** status

HAL_HCD_HC_Halt

Function name	**HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)**
Function description	Halt a host channel.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15
Return values	• **HAL:** status

HAL_HCD_MspInit

Function name	**void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)**
Function description	Initializes the HCD MSP.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_MspDeInit

Function name	**void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)**

Function description	DeInitializes HCD MSP.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_HC_SubmitRequest

Function name	**HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)**
Function description	Submit a new URB for processing.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15 • **direction:** Channel number. This parameter can be one of these values: 0 : Output / 1 : Input • **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/ • **token:** Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1 • **pbuff:** pointer to URB data • **length:** Length of URB data • **do_ping:** activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active
Return values	• **HAL:** status

HAL_HCD_IRQHandler

Function name	**void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)**
Function description	handle HCD interrupt request.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_SOF_Callback

Function name	**void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)**
Function description	SOF callback.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_Connect_Callback

Function name	**void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)**

Function description	Connexion Event callback.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_Disconnect_Callback

Function name	**void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)**
Function description	Disonnection Event callback.
Parameters	• **hhcd:** HCD handle
Return values	• **None:**

HAL_HCD_HC_NotifyURBChange_Callback

Function name	**void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)**
Function description	Notify URB state change callback.
Parameters	• **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	• **None:**

HAL_HCD_ResetPort

Function name	**HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)**
Function description	Reset the host port.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_Start

Function name	**HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)**
Function description	Start the host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_Stop

Function name	**HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)**

Function description	Stop the host driver.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** status

HAL_HCD_GetState

Function name	**HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)**
Function description	Return the HCD handle state.
Parameters	• **hhcd:** HCD handle
Return values	• **HAL:** state

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	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15
Return values	• **URB:** state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

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	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15
Return values	• **last:** transfer size in byte

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	• **hhcd:** HCD handle • **chnum:** Channel number. This parameter can be a value from 1 to 15
Return values	• **Host:** channel state This parameter can be one of the these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR/

HAL_HCD_GetCurrentFrame

Function name	**uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)**
Function description	Return the current Host frame number.
Parameters	• **hhcd:** HCD handle
Return values	• **Current:** Host frame number

HAL_HCD_GetCurrentSpeed

Function name	**uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)**
Function description	Return the Host enumeration speed.
Parameters	• **hhcd:** HCD handle
Return values	• **Enumeration:** speed

22.3 HCD Firmware driver defines

22.3.1 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 Speed

HCD_SPEED_LOW

HCD_SPEED_FULL

23 HAL I2C Generic Driver

23.1 I2C Firmware driver registers structures

23.1.1 I2C_InitTypeDef

Data Fields

- ***uint32_t ClockSpeed***
- ***uint32_t DutyCycle***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::ClockSpeed***
  Specifies the clock frequency. This parameter must be set to a value lower than 400kHz
- ***uint32_t I2C_InitTypeDef::DutyCycle***
  Specifies the I2C fast mode duty cycle. This parameter can be a value of ***I2C_duty_cycle_in_fast_mode***
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_addressing_mode***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_dual_addressing_mode***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_general_call_addressing_mode***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_nostretch_mode***

23.1.2 I2C_HandleTypeDef

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***

- ***__IO uint32_t PreviousState***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO HAL_I2C_ModeTypeDef Mode***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Devaddress***
- ***__IO uint32_t Memaddress***
- ***__IO uint32_t MemaddSize***
- ***__IO uint32_t EventCount***

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
- ***__IO uint32_t I2C_HandleTypeDef::XferOptions***
  I2C transfer options
- ***__IO uint32_t I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state and mode context for internal usage
- ***DMA_HandleTypeDef* I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t I2C_HandleTypeDef::Devaddress***
  I2C Target device address
- ***__IO uint32_t I2C_HandleTypeDef::Memaddress***
  I2C Target memory address
- ***__IO uint32_t I2C_HandleTypeDef::MemaddSize***
  I2C Target memory address size
- ***__IO uint32_t I2C_HandleTypeDef::EventCount***
  I2C Event counter

23.2 I2C Firmware driver API description

23.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implementing 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 channel
      – Enable the DMAx interface clock using
      – Configure the DMA handle parameters
      – Configure the DMA Tx or Rx channel
      – Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

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

Polling mode IO MEM operation

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

Interrupt mode IO operation

- Transmit in master mode an amount of data in non blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback

- Receive in master mode an amount of data in non blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()

Interrupt mode IO sequential operation



These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer
- Option field values are defined through @ref I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functionnal is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
- Differents sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Sequential_Transmit_IT()
    - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Sequential_Receive_IT()
    - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()

  - Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
    - End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
    - When address slave I2C match, HAL_I2C_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Sequential_Transmit_IT()
    - At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Sequential_Receive_IT()
    - At reception end of current frame 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
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()

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

I2C Workarounds linked to Silicon Limitation



See ErrataSheet to know full silicon limitation list of your product. (#) Workarounds Implemented inside I2C HAL Driver (##) Wrong data read into data register (Polling and Interrupt mode) (##) Start cannot be generated after a misplaced Stop (##) Some software events must be managed before the current byte is being transferred: Workaround: Use DMA in general, except when the Master is receiving a single byte. For Interupt mode, I2C should have the highest priority in the application. (##) Mismatch on the "Setup time for a repeated Start condition" timing parameter: Workaround: Reduce the frequency down to 88 kHz or use the I2C Fast-mode if supported by the slave. (##) Data valid time

(tVD;DAT) violated without the OVR flag being set: Workaround: If the slave device allows it, use the clock stretching mechanism by programming NoStretchMode = I2C_NOSTRETCH_DISABLE in HAL_I2C_Init.

23.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
  - Communication Speed
  - Duty cycle
  - Addressing mode
  - Own Address 1
  - Dual Addressing mode
  - Own Address 2
  - General call mode
  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- ***HAL_I2C_Init()***
- ***HAL_I2C_DeInit()***
- ***HAL_I2C_MspInit()***
- ***HAL_I2C_MspDeInit()***

23.2.3 IO operation functions

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

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2C_Master_Transmit()
   - HAL_I2C_Master_Receive()
   - HAL_I2C_Slave_Transmit()
   - HAL_I2C_Slave_Receive()
   - HAL_I2C_Mem_Write()
   - HAL_I2C_Mem_Read()
   - HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2C_Master_Transmit_IT()
   - HAL_I2C_Master_Receive_IT()
   - HAL_I2C_Slave_Transmit_IT()
   - HAL_I2C_Slave_Receive_IT()

  - HAL_I2C_Master_Sequential_Transmit_IT()
  - HAL_I2C_Master_Sequential_Receive_IT()
  - HAL_I2C_Slave_Sequential_Transmit_IT()
  - HAL_I2C_Slave_Sequential_Receive_IT()
  - HAL_I2C_Mem_Write_IT()
  - HAL_I2C_Mem_Read_IT()
4. No-Blocking mode functions with DMA are :
  - HAL_I2C_Master_Transmit_DMA()
  - HAL_I2C_Master_Receive_DMA()
  - HAL_I2C_Slave_Transmit_DMA()
  - HAL_I2C_Slave_Receive_DMA()
  - HAL_I2C_Mem_Write_DMA()
  - HAL_I2C_Mem_Read_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_I2C_MemTxCpltCallback()
  - HAL_I2C_MemRxCpltCallback()
  - HAL_I2C_MasterTxCpltCallback()
  - HAL_I2C_MasterRxCpltCallback()
  - HAL_I2C_SlaveTxCpltCallback()
  - HAL_I2C_SlaveRxCpltCallback()
  - HAL_I2C_ErrorCallback()
  - HAL_I2C_AbortCpltCallback()

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_Master_Sequential_Transmit_IT()***
- ***HAL_I2C_Master_Sequential_Receive_IT()***
- ***HAL_I2C_Slave_Transmit_IT()***
- ***HAL_I2C_Slave_Receive_IT()***
- ***HAL_I2C_Slave_Sequential_Transmit_IT()***
- ***HAL_I2C_Slave_Sequential_Receive_IT()***
- ***HAL_I2C_EnableListen_IT()***
- ***HAL_I2C_DisableListen_IT()***
- ***HAL_I2C_Master_Transmit_DMA()***
- ***HAL_I2C_Master_Receive_DMA()***
- ***HAL_I2C_Master_Abort_IT()***
- ***HAL_I2C_Slave_Transmit_DMA()***
- ***HAL_I2C_Slave_Receive_DMA()***
- ***HAL_I2C_Mem_Write()***
- ***HAL_I2C_Mem_Read()***
- ***HAL_I2C_Mem_Write_IT()***
- ***HAL_I2C_Mem_Read_IT()***
- ***HAL_I2C_Mem_Write_DMA()***
- ***HAL_I2C_Mem_Read_DMA()***
- ***HAL_I2C_IsDeviceReady()***
- ***HAL_I2C_EV_IRQHandler()***
- ***HAL_I2C_ER_IRQHandler()***

- ***HAL_I2C_MasterTxCpltCallback()***
- ***HAL_I2C_MasterRxCpltCallback()***
- ***HAL_I2C_SlaveTxCpltCallback()***
- ***HAL_I2C_SlaveRxCpltCallback()***
- ***HAL_I2C_AddrCallback()***
- ***HAL_I2C_ListenCpltCallback()***
- ***HAL_I2C_MemTxCpltCallback()***
- ***HAL_I2C_MemRxCpltCallback()***
- ***HAL_I2C_ErrorCallback()***
- ***HAL_I2C_AbortCpltCallback()***

23.2.4 Peripheral State, Mode and Error functions

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

This section contains the following APIs:

- ***HAL_I2C_GetState()***
- ***HAL_I2C_GetMode()***
- ***HAL_I2C_GetError()***

23.2.5 Detailed description of functions

HAL_I2C_Init

Function name	**HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)**
Function description	Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and create the associated handle.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• **HAL:** status

HAL_I2C_DeInit

Function name	**HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)**
Function description	DeInitializes the I2C peripheral.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• **HAL:** status

HAL_I2C_MspInit

Function name	**void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)**
Function description	I2C MSP Init.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• **None:**

HAL_I2C_MspDeInit

Function name **void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)**

Function description I2C MSP DeInit.

Parameters
- **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values
- **None:**

HAL_I2C_Master_Transmit

Function name **HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Transmits in master mode an amount of data in blocking mode.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_I2C_Master_Receive

Function name **HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Receives in master mode an amount of data in blocking mode.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_I2C_Slave_Transmit

Function name **HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Transmits in slave mode an amount of data in blocking mode.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer

- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_I2C_Slave_Receive

Function name **HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Receive in slave mode an amount of data in blocking mode.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_I2C_Mem_Write

Function name **HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Write an amount of data in blocking mode to a specific memory address.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_I2C_Mem_Read

Function name **HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Read an amount of data in blocking mode from a specific memory address.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer

- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_I2C_IsDeviceReady

Function name
HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description
Checks if target device is ready for communication.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values
- **HAL:** status

Notes
- This function is used with Memory devices

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 non-blocking mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_I2C_Mem_Write_IT

Function name **HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**

Function description Write an amount of data in non-blocking mode with Interrupt to a specific memory address.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_I2C_Mem_Read_IT

Function name **HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**

Function description: Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values:
- **HAL:** status

HAL_I2C_Master_Sequential_Transmit_IT

Function name: **HAL_StatusTypeDef HAL_I2C_Master_Sequential_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**

Function description: Sequential transmit in master mode an amount of data in non-blocking mode with Interrupt.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C XferOptions definition

Return values:
- **HAL:** status

Notes:
- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Sequential_Receive_IT

Function name: **HAL_StatusTypeDef HAL_I2C_Master_Sequential_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**

Function description: Sequential receive in master mode an amount of data in non-blocking mode with Interrupt.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C XferOptions definition

Return values:
- **HAL:** status

Notes
- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Transmit_IT

Function name **HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**

Function description Sequential transmit in slave mode an amount of data in no-blocking mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C XferOptions definition

Return values
- **HAL:** status

Notes
- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Receive_IT

Function name **HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**

Function description Sequential receive in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C XferOptions definition

Return values
- **HAL:** status

Notes
- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Abort_IT

Function name **HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)**

Function description Abort a master I2C process communication with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface

Return values
- **HAL:** status

Notes
- This abort can be called only if state is ready

HAL_I2C_EnableListen_IT

Function name **HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)**

Function description Enable the Address listen mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **HAL:** status

HAL_I2C_DisableListen_IT

Function name **HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)**

Function description Disable the Address listen mode with Interrupt.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **HAL:** status

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 non-blocking mode with DMA.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with DMA.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shift at right before call interface

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with DMA.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with DMA.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name **HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)**

Function description Write an amount of data in non-blocking mode with DMA to a specific memory address.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

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 non-blocking mode with DMA from a specific memory address.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be read

Return values:
- **HAL:** status

HAL_I2C_EV_IRQHandler

Function name: **void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)**

Function description: This function handles I2C event interrupt request.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values:
- **None:**

HAL_I2C_ER_IRQHandler

Function name: **void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)**

Function description: This function handles I2C error interrupt request.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values:
- **None:**

HAL_I2C_MasterTxCpltCallback

Function name: **void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description: Master Tx Transfer completed callback.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values:
- **None:**

HAL_I2C_MasterRxCpltCallback

Function name: **void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description: Master Rx Transfer completed callback.

Parameters:
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_SlaveTxCpltCallback

Function name **void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description Slave Tx Transfer completed callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_SlaveRxCpltCallback

Function name **void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description Slave Rx Transfer completed callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_AddrCallback

Function name **void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)**

Function description Slave Address Match callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of I2C XferOptions definition
- **AddrMatchCode:** Address Match Code

Return values
- **None:**

HAL_I2C_ListenCpltCallback

Function name **void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description Listen Complete callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_MemTxCpltCallback

Function name **void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description Memory Tx Transfer completed callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains

the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_MemRxCpltCallback

Function name **void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description Memory Rx Transfer completed callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_ErrorCallback

Function name **void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)**

Function description I2C error callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_AbortCpltCallback

Function name **void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)**

Function description I2C abort callback.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **None:**

HAL_I2C_GetState

Function name **HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)**

Function description Return the I2C handle state.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values
- **HAL:** state

HAL_I2C_GetMode

Function name **HAL_I2C_ModeTypeDef HAL_I2C_GetMode (I2C_HandleTypeDef * hi2c)**

Function description Return the I2C Master, Slave, Memory or no mode.

Parameters
- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values
- **HAL:** mode

HAL_I2C_GetError

Function name	**uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)**
Function description	Return the I2C error code.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• **I2C:** Error Code

23.3 I2C Firmware driver defines

23.3.1 I2C

I2C addressing mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C dual addressing mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C duty cycle in fast mode

I2C_DUTYCYCLE_2

I2C_DUTYCYCLE_16_9

I2C Error Code

HAL_I2C_ERROR_NONE	No error
HAL_I2C_ERROR_BERR	BERR error
HAL_I2C_ERROR_ARLO	ARLO error
HAL_I2C_ERROR_AF	AF error
HAL_I2C_ERROR_OVR	OVR error
HAL_I2C_ERROR_DMA	DMA transfer error
HAL_I2C_ERROR_TIMEOUT	Timeout Error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable or disable the specified I2C

interrupts.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2C_IT_BUF: Buffer interrupt enable
  - I2C_IT_EVT: Event interrupt enable
  - I2C_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

__HAL_I2C_GET_IT_SOURCE

Description:

- Checks if the specified I2C interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_BUF: Buffer interrupt enable
  - I2C_IT_EVT: Event interrupt enable
  - I2C_IT_ERR: Error interrupt enable

Return value:

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

__HAL_I2C_GET_FLAG

Description:

- Checks whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_SMBALERT: SMBus Alert flag
  - I2C_FLAG_TIMEOUT: Timeout or Tlow error flag
  - I2C_FLAG_PECERR: PEC error in reception flag

- I2C_FLAG_OVR: Overrun/Underrun flag
- I2C_FLAG_AF: Acknowledge failure flag
- I2C_FLAG_ARLO: Arbitration lost flag
- I2C_FLAG_BERR: Bus error flag
- I2C_FLAG_TXE: Data register empty flag
- I2C_FLAG_RXNE: Data register not empty flag
- I2C_FLAG_STOPF: Stop detection flag
- I2C_FLAG_ADD10: 10-bit header sent flag
- I2C_FLAG_BTF: Byte transfer finished flag
- I2C_FLAG_ADDR: Address sent flag Address matched flag
- I2C_FLAG_SB: Start bit flag
- I2C_FLAG_DUALF: Dual flag
- I2C_FLAG_SMBHOST: SMBus host header
- I2C_FLAG_SMBDEFAULT: SMBus default header
- I2C_FLAG_GENCALL: General call header flag
- I2C_FLAG_TRA: Transmitter/Receiver flag
- I2C_FLAG_BUSY: Bus busy flag
- I2C_FLAG_MSL: Master/Slave flag

Return value:

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

__HAL_I2C_CLEAR_FLAG

Description:

- Clears the I2C pending flags which are cleared by writing 0 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_SMBALERT: SMBus Alert flag
  - I2C_FLAG_TIMEOUT: Timeout or Tlow error flag
  - I2C_FLAG_PECERR: PEC error in reception flag
  - I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode)
  - I2C_FLAG_AF: Acknowledge failure

flag
  - I2C_FLAG_ARLO: Arbitration lost flag (Master mode)
  - I2C_FLAG_BERR: Bus error flag

Return value:

- None

__HAL_I2C_CLEAR_ADDRFLAG

Description:

- Clears the I2C ADDR pending flag.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

Return value:

- None

__HAL_I2C_CLEAR_STOPFLAG

Description:

- Clears the I2C STOPF pending flag.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.

Return value:

- None

I2C Flag definition

I2C_FLAG_SMBALERT

I2C_FLAG_TIMEOUT

I2C_FLAG_PECERR

I2C_FLAG_OVR

I2C_FLAG_AF

I2C_FLAG_ARLO

I2C_FLAG_BERR

I2C_FLAG_TXE

I2C_FLAG_RXNE

I2C_FLAG_STOPF

I2C_FLAG_ADD10

I2C_FLAG_BTF

I2C_FLAG_ADDR

I2C_FLAG_SB

I2C_FLAG_DUALF

I2C_FLAG_SMBHOST

I2C_FLAG_SMBDEFAULT

I2C_FLAG_GENCALL

I2C_FLAG_TRA

I2C_FLAG_BUSY

I2C_FLAG_MSL

I2C general call addressing mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_BUF

I2C_IT_EVT

I2C_IT_ERR

I2C Private macros to check input parameters

IS_I2C_DUTY_CYCLE

IS_I2C_ADDRESSING_MODE

IS_I2C_DUAL_ADDRESS

IS_I2C_GENERAL_CALL

IS_I2C_NO_STRETCH

IS_I2C_MEMADD_SIZE

IS_I2C_CLOCK_SPEED

IS_I2C_OWN_ADDRESS1

IS_I2C_OWN_ADDRESS2

IS_I2C_TRANSFER_OPTIONS_REQUEST

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C nostretch mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C XferDirection definition

I2C_DIRECTION_RECEIVE

I2C_DIRECTION_TRANSMIT

I2C XferOptions definition

I2C_FIRST_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

24 HAL I2S Generic Driver

24.1 I2S Firmware driver registers structures

24.1.1 I2S_InitTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***

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***

24.1.2 __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***
- ***void(* IrqHandlerISR***
- ***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 registers base address
- ***I2S_InitTypeDef __I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***uint16_t* __I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* __I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***void(* __I2S_HandleTypeDef::IrqHandlerISR)(struct __I2S_HandleTypeDef *hi2s)***
  I2S function pointer on IrqHandler
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters
- ***__IO HAL_LockTypeDef __I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef __I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO uint32_t __I2S_HandleTypeDef::ErrorCode***
  I2S Error code This parameter can be a value of ***I2S_ErrorCode***

24.2 I2S Firmware driver API description

24.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      – Enable the clock for the I2S GPIOs.
      – Configure these I2S pins as alternate function.
   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. The I2SxCLK source is the system clock (provided by the HSI, the HSE or the PLL, and sourcing the AHB clock). For connectivity line devices, the I2SxCLK source can be either SYSCLK or the PLL3 VCO (2 x PLL3CLK) clock in order to achieve the maximum accuracy. Make sure that either: External clock source is configured after setting correctly the define constant HSE_VALUE in the stm32f1xx_hal_conf.h file.
4. Three operation modes are available within this driver :

Polling mode IO operation

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

Interrupt mode IO operation

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

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback

- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in 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

I2C Workarounds linked to Silicon Limitation



Only the 16-bit mode with no data extension can be used when the I2S is in Master and used the PCM long synchronization mode.

24.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function HAL_I2S_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()***

24.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2S_Transmit()
   - HAL_I2S_Receive()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2S_Transmit_IT()
   - HAL_I2S_Receive_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2S_Transmit_DMA()
   - HAL_I2S_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2S_TxCpltCallback()
   - HAL_I2S_RxCpltCallback()
   - HAL_I2S_ErrorCallback()

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()***

24.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- ***HAL_I2S_GetState()***
- ***HAL_I2S_GetError()***

24.2.5 Detailed description of functions

HAL_I2S_Init

Function name	**HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)**
Function description	Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **HAL:** status

HAL_I2S_DeInit

Function name **HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)**

Function description DeInitializes the I2S peripheral.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **HAL:** status

HAL_I2S_MspInit

Function name **void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)**

Function description I2S MSP Init.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_MspDeInit

Function name **void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)**

Function description I2S MSP DeInit.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_Transmit

Function name **HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Transmit an amount of data in blocking mode.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values
- **HAL:** status

Notes
- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the

clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive

Function name	**HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to data buffer • **Size:** number of data sample to be sent: • **Timeout:** Timeout duration
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming) • In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction

HAL_I2S_Transmit_IT

Function name	**HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to data buffer. • **Size:** number of data sample to be sent:
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_IT

Function name	**HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)**
Function description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to the Receive data buffer. • **Size:** number of data sample to be sent:
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). • It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.

HAL_I2S_IRQHandler

Function name	**void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)**
Function description	This function handles I2S interrupt request.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **None:**

HAL_I2S_Transmit_DMA

Function name	**HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)**
Function description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to the Transmit data buffer. • **Size:** number of data sample to be sent:
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example:

audio streaming).

HAL_I2S_Receive_DMA

Function name	**HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)**
Function description	Receive an amount of data in non-blocking mode with DMA.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • **pData:** a 16-bit pointer to the Receive data buffer. • **Size:** number of data sample to be sent:
Return values	• **HAL:** status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_DMAPause

Function name	**HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)**
Function description	Pauses the audio channel playing from the Media.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL:** status

HAL_I2S_DMAResume

Function name	**HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)**
Function description	Resumes the audio channel playing from the Media.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL:** status

HAL_I2S_DMAStop

Function name	**HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)**
Function description	Resumes the audio channel playing from the Media.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **HAL:** status

HAL_I2S_TxHalfCpltCallback

Function name **void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)**

Function description Tx Transfer Half completed callbacks.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_TxCpltCallback

Function name **void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)**

Function description Tx Transfer completed callbacks.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_RxHalfCpltCallback

Function name **void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)**

Function description Rx Transfer half completed callbacks.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_RxCpltCallback

Function name **void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)**

Function description Rx Transfer completed callbacks.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_ErrorCallback

Function name **void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)**

Function description I2S error callbacks.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **None:**

HAL_I2S_GetState

Function name **HAL_I2S_StateTypeDef HAL_I2S_GetState**

(I2S_HandleTypeDef * hi2s)

Function description — Return the I2S state.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **HAL:** state

HAL_I2S_GetError

Function name — **uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)**

Function description — Return the I2S error code.

Parameters
- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values
- **I2S:** Error Code

24.3 I2S Firmware driver defines

24.3.1 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 Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error Code

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_PRESCALER	Prescaler Calculation error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable 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

Description:

- 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_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 Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

25 HAL IRDA Generic Driver

25.1 IRDA Firmware driver registers structures

25.1.1 IRDA_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint32_t IrDAMode***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
  This member configures the IRDA communication baud rate. The baud rate is computed using the following formula:IntegerDivider = ((PCLKx) / (16 * (hirda->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t IRDA_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDA_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***
  Specifies the Prescaler value prescaler value to be programmed in the IrDA low-power Baud Register, for defining pulse width on which burst acceptance/rejection will be decided. This value is used as divisor of system clock to achieve required pulse width.
- ***uint32_t IRDA_InitTypeDef::IrDAMode***
  Specifies the IrDA mode This parameter can be a value of ***IRDA_Low_Power***

25.1.2 IRDA_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***

- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO uint32_t IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code

25.2 IRDA Firmware driver API description

25.2.1 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 the 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 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 IRDA DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
  - Configure the IRDAx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)

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.

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 half transfer HAL_IRDA_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback
- 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 half transfer HAL_IRDA_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback

- 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
- Pause the DMA Transfer using HAL_IRDA_DMAPause()
- Resume the DMA Transfer using HAL_IRDA_DMAResume()
- Stop the DMA Transfer using HAL_IRDA_DMAStop()

IRDA HAL driver macros list

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

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether the specified IRDA interrupt has occurred or not



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



Additional remark: 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. The IRDA frame formats depend on the frame length defined by the M bit (8-bits or 9-bits). Refer to the product reference manual for details.

25.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in IrDA mode.

- For the asynchronous mode only these parameters can be configured:
  - BaudRate
  - WordLength
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible IRDA frame formats.
  - Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
  - Mode: Receiver/transmitter modes
  - IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

The HAL_IRDA_Init() API follows IRDA configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***

25.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers. IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are:
   - HAL_IRDA_Transmit()
   - HAL_IRDA_Receive()
3. Non Blocking mode APIs with Interrupt are:
   - HAL_IRDA_Transmit_IT()
   - HAL_IRDA_Receive_IT()
   - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are:
   - HAL_IRDA_Transmit_DMA()
   - HAL_IRDA_Receive_DMA()
   - HAL_IRDA_DMAPause()
   - HAL_IRDA_DMAResume()
   - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_IRDA_TxHalfCpltCallback()
   - HAL_IRDA_TxCpltCallback()
   - HAL_IRDA_RxHalfCpltCallback()
   - HAL_IRDA_RxCpltCallback()
   - HAL_IRDA_ErrorCallback()

This section contains the following APIs:

- ***HAL_IRDA_Transmit()***
- ***HAL_IRDA_Receive()***
- ***HAL_IRDA_Transmit_IT()***
- ***HAL_IRDA_Receive_IT()***
- ***HAL_IRDA_Transmit_DMA()***
- ***HAL_IRDA_Receive_DMA()***
- ***HAL_IRDA_DMAPause()***
- ***HAL_IRDA_DMAResume()***
- ***HAL_IRDA_DMAStop()***

- ***HAL_IRDA_Abort()***
- ***HAL_IRDA_AbortTransmit()***
- ***HAL_IRDA_AbortReceive()***
- ***HAL_IRDA_Abort_IT()***
- ***HAL_IRDA_AbortTransmit_IT()***
- ***HAL_IRDA_AbortReceive_IT()***
- ***HAL_IRDA_IRQHandler()***
- ***HAL_IRDA_TxCpltCallback()***
- ***HAL_IRDA_TxHalfCpltCallback()***
- ***HAL_IRDA_RxCpltCallback()***
- ***HAL_IRDA_RxHalfCpltCallback()***
- ***HAL_IRDA_ErrorCallback()***
- ***HAL_IRDA_AbortCpltCallback()***
- ***HAL_IRDA_AbortTransmitCpltCallback()***
- ***HAL_IRDA_AbortReceiveCpltCallback()***

25.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IrDA peripheral.
- HAL_IRDA_GetError() check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- ***HAL_IRDA_GetState()***
- ***HAL_IRDA_GetError()***

25.2.5 Detailed description of functions

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	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL:** status

HAL_IRDA_DeInit

Function name	**HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)**
Function description	DeInitializes the IRDA peripheral.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **HAL:** status

HAL_IRDA_MspInit

Function name **void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)**

Function description IRDA MSP Init.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **None:**

HAL_IRDA_MspDeInit

Function name **void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)**

Function description IRDA MSP DeInit.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **None:**

HAL_IRDA_Transmit

Function name **HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Sends an amount of data in blocking mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Specify timeout value

Return values
- **HAL:** status

HAL_IRDA_Receive

Function name **HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Receive an amount of data in blocking mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Specify timeout value

Return values
- **HAL:** status

HAL_IRDA_Transmit_IT

Function name **HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**

Function description Sends an amount of data in non blocking mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_IRDA_Receive_IT

Function name **HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**

Function description Receives an amount of data in non blocking mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values
- **HAL:** status

HAL_IRDA_Transmit_DMA

Function name **HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**

Function description Sends an amount of data in non blocking mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_IRDA_Receive_DMA

Function name **HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**

Function description Receives an amount of data in non blocking mode.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values
- **HAL:** status

Notes
- When the IRDA parity is enabled (PCE = 1) the data received contain the parity bit.

HAL_IRDA_DMAPause

Function name **HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)**

Function description Pauses the DMA Transfer.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **HAL:** status

HAL_IRDA_DMAResume

Function name **HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)**

Function description Resumes the DMA Transfer.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **HAL:** status

HAL_IRDA_DMAStop

Function name **HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)**

Function description Stops the DMA Transfer.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **HAL:** status

HAL_IRDA_Abort

Function name **HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)**

Function description Abort ongoing transfers (blocking mode).

Parameters
- **hirda:** IRDA handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting

function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Transmit transfer (blocking mode).
Parameters	• **hirda:** IRDA handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Receive transfer (blocking mode).
Parameters	• **hirda:** IRDA handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY • This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **hirda:** IRDA handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback

- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **hirda:** IRDA handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **hirda:** IRDA handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name	**void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)**
Function description	This function handles IRDA interrupt request.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA

module.

Return values
- **None:**

HAL_IRDA_TxCpltCallback

Function name **void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)**

Function description Tx Transfer complete callbacks.

Parameters
- **hirda:** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **None:**

HAL_IRDA_RxCpltCallback

Function name **void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)**

Function description Rx Transfer complete callbacks.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **None:**

HAL_IRDA_TxHalfCpltCallback

Function name **void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**

Function description Tx Half Transfer completed callbacks.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_IRDA_RxHalfCpltCallback

Function name **void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**

Function description Rx Half Transfer complete callbacks.

Parameters
- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values
- **None:**

HAL_IRDA_ErrorCallback

Function name **void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)**

Function description	IRDA error callbacks.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **None:**

HAL_IRDA_AbortCpltCallback

Function name	**void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Complete callback.
Parameters	• **hirda:** IRDA handle.
Return values	• **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name	**void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort Transmit Complete callback.
Parameters	• **hirda:** IRDA handle.
Return values	• **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name	**void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)**
Function description	IRDA Abort ReceiveComplete callback.
Parameters	• **hirda:** IRDA handle.
Return values	• **None:**

HAL_IRDA_GetState

Function name	**HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)**
Function description	Returns the IRDA state.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL:** state

HAL_IRDA_GetError

Function name	**uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)**
Function description	Return the IRDA error code.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA

module.

Return values
- **IRDA:** Error Code

25.3 IRDA Firmware driver defines

25.3.1 IRDA

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

__HAL_IRDA_RESET_HANDLE_STATE

Description:
- Reset IRDA handle gstate & RxState.

Parameters:
- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_FLUSH_DRREGISTER

Description:
- Flush the IRDA DR register.

Parameters:
- __HANDLE__: specifies the USART Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_GET_FLAG

Description:
- Check whether the specified IRDA flag is set or not.

Parameters:
- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – IRDA_FLAG_TXE: Transmit data

register empty flag
  - IRDA_FLAG_TC: Transmission Complete flag
  - IRDA_FLAG_RXNE: Receive data register not empty flag
  - IRDA_FLAG_IDLE: Idle Line detection flag
  - IRDA_FLAG_ORE: OverRun Error flag
  - IRDA_FLAG_NE: Noise Error flag
  - IRDA_FLAG_FE: Framing Error flag
  - IRDA_FLAG_PE: Parity Error flag

Return value:

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

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_FLAG_TC: Transmission Complete flag.
  - IRDA_FLAG_RXNE: Receive data register not empty flag.

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

- __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)

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __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)

__HAL_IRDA_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __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_ERR: Error interrupt
- IRDA_IT_PE: Parity Error interrupt

Return value:

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

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

IRDA Flags

IRDA_FLAG_TXE

IRDA_FLAG_TC

IRDA_FLAG_RXNE

IRDA_FLAG_IDLE

IRDA_FLAG_ORE

IRDA_FLAG_NE

IRDA_FLAG_FE

IRDA_FLAG_PE

IRDA Interrupt Definitions

IRDA_IT_PE

IRDA_IT_TXE

IRDA_IT_TC

IRDA_IT_RXNE

IRDA_IT_IDLE

IRDA_IT_LBD

IRDA_IT_CTS

IRDA_IT_ERR

IRDA Low Power

IRDA_POWERMODE_LOWPOWER

IRDA_POWERMODE_NORMAL

IRDA Parity

IRDA_PARITY_NONE

IRDA_PARITY_EVEN

IRDA_PARITY_ODD

IRDA Transfer Mode

IRDA_MODE_RX

IRDA_MODE_TX

IRDA_MODE_TX_RX

IRDA Word Length

IRDA_WORDLENGTH_8B

IRDA_WORDLENGTH_9B

26 HAL IWDG Generic Driver

26.1 IWDG Firmware driver registers structures

26.1.1 IWDG_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

26.1.2 IWDG_HandleTypeDef

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
  IWDG required parameters

26.2 IWDG Firmware driver API description

26.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by Low-Speed clock (LSI) and thus stays active even if the main clock fails.
- Once the IWDG is started, the LSI is forced ON and both can not be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode : When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros

Min-max timeout value @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI frequency dispersion. STM32F1xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM5 CH4 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

26.2.2 How to use this driver

1. Use IWDG using HAL_IWDG_Init() function to :
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts downcounting.
   - Enable write access to configuration register: IWDG_PR & IWDG_RLR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - wait for status flags to be reset"
2. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

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

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

26.2.3 Initialization and Start functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:

- ***HAL_IWDG_Init()***

26.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh()***

26.2.5 Detailed description of functions

HAL_IWDG_Init

Function name	**HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)**
Function description	Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.

Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL:** status

HAL_IWDG_Refresh

Function name	**HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)**
Function description	Refresh the IWDG.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL:** status

26.3 IWDG Firmware driver defines

26.3.1 IWDG

IWDG Exported Macros

__HAL_IWDG_START	**Description:** • Enable the IWDG peripheral. **Parameters:** • __HANDLE__: IWDG handle **Return value:** • None
__HAL_IWDG_RELOAD_COUNTER	**Description:** • Reload IWDG counter with value defined in the reload register (write access to IWDG_PR & IWDG_RLR registers disabled). **Parameters:** • __HANDLE__: IWDG handle **Return value:** • None

IWDG Prescaler

IWDG_PRESCALER_4	IWDG prescaler set to 4
IWDG_PRESCALER_8	IWDG prescaler set to 8
IWDG_PRESCALER_16	IWDG prescaler set to 16
IWDG_PRESCALER_32	IWDG prescaler set to 32
IWDG_PRESCALER_64	IWDG prescaler set to 64
IWDG_PRESCALER_128	IWDG prescaler set to 128

IWDG_PRESCALER_256 IWDG prescaler set to 256

27 HAL NAND Generic Driver

27.1 NAND Firmware driver registers structures

27.1.1 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***

27.1.2 NAND_AddressTypeDef

Data Fields

- ***uint16_t Page***
- ***uint16_t Plane***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Plane***
  NAND memory Plane address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

27.1.3 NAND_DeviceConfigTypeDef

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t PlaneNbr***
- ***uint32_t PlaneSize***
- ***FunctionalState ExtraCommandEnable***

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
  NAND memory page (without spare area) size measured in bytes for 8 bits adressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits adressing or words for 16 bits addressing

- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages
- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory plane size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
  NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. Example: Toshiba THTH58BYG3S0HBAI6. This parameter could be ENABLE or DISABLE Please check the Read Mode sequnece in the NAND device datasheet

27.1.4 NAND_HandleTypeDef

Data Fields

- ***FSMC_NAND_TypeDef * Instance***
- ***FSMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***

Field Documentation

- ***FSMC_NAND_TypeDef* NAND_HandleTypeDef::Instance***
  Register base address
- ***FSMC_NAND_InitTypeDef NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_DeviceConfigTypeDef NAND_HandleTypeDef::Config***
  NAND phusical characteristic information structure

27.2 NAND Firmware driver API description

27.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FSMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page

size..) predefined by the user in the NAND_DeviceConfigTypeDef 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.

27.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- ***HAL_NAND_Init()***
- ***HAL_NAND_DeInit()***
- ***HAL_NAND_MspInit()***
- ***HAL_NAND_MspDeInit()***
- ***HAL_NAND_IRQHandler()***
- ***HAL_NAND_ITCallback()***
- ***HAL_NAND_ConfigDevice()***
- ***HAL_NAND_Read_ID()***

27.2.3 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_ConfigDevice()***
- ***HAL_NAND_Read_Page_8b()***
- ***HAL_NAND_Read_Page_16b()***
- ***HAL_NAND_Write_Page_8b()***
- ***HAL_NAND_Write_Page_16b()***
- ***HAL_NAND_Read_SpareArea_8b()***
- ***HAL_NAND_Read_SpareArea_16b()***
- ***HAL_NAND_Write_SpareArea_8b()***
- ***HAL_NAND_Write_SpareArea_16b()***
- ***HAL_NAND_Erase_Block()***
- ***HAL_NAND_Read_Status()***
- ***HAL_NAND_Address_Inc()***

27.2.4 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()***

27.2.5 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()***

27.2.6 Detailed description of functions

HAL_NAND_Init

Function name	**HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FSMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FSMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)**
Function description	Perform NAND memory Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **ComSpace_Timing:** pointer to Common space timing structure • **AttSpace_Timing:** pointer to Attribute space timing structure
Return values	• **HAL:** status

HAL_NAND_DeInit

Function name	**HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)**
Function description	Perform NAND memory De-Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• **HAL:** status

HAL_NAND_ConfigDevice

Function name	**HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, NAND_DeviceConfigTypeDef * pDeviceConfig)**
Function description	Configure the device: Enter the physical parameters of the device.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that

contains the configuration information for NAND module.
- **pDeviceConfig:** : pointer to NAND_DeviceConfigTypeDef structure

Return values
- **HAL:** status

HAL_NAND_Read_ID

Function name **HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)**

Function description Read the NAND memory electronic signature.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values
- **HAL:** status

HAL_NAND_MspInit

Function name **void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)**

Function description NAND MSP Init.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **None:**

HAL_NAND_MspDeInit

Function name **void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)**

Function description NAND MSP DeInit.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **None:**

HAL_NAND_IRQHandler

Function name **void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)**

Function description This function handles NAND device interrupt request.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_ITCallback

Function name **void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)**

Function description NAND interrupt feature callback.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **None:**

HAL_NAND_Reset

Function name **HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)**

Function description NAND memory reset.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_Read_Page_8b

Function name **HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)**

Function description Read Page(s) from NAND memory block (8-bits addressing)

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to destination read buffer
- **NumPageToRead:** : number of pages to read from block

Return values
- **HAL:** status

HAL_NAND_Write_Page_8b

Function name **HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)**

Function description Write Page(s) to NAND memory block (8-bits addressing)

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write
- **NumPageToWrite:** : number of pages to write to block

Return values
- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name **HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)**

Function description Read Spare area(s) from NAND memory (8-bits addressing)

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write

- **NumSpareAreaToRead:** Number of spare area to read

Return values
- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name **HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)**

Function description Write Spare area(s) to NAND memory (8-bits addressing)

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write
- **NumSpareAreaTowrite:** : number of spare areas to write to block

Return values
- **HAL:** status

HAL_NAND_Read_Page_16b

Function name **HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)**

Function description Read Page(s) from NAND memory block (16-bits addressing)

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** : number of pages to read from block

Return values
- **HAL:** status

HAL_NAND_Write_Page_16b

Function name **HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToWrite)**

Function description Write Page(s) to NAND memory block (16-bits addressing)

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** : number of pages to write to block

Return values
- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name **HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef ***

pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description: Read Spare area(s) from NAND memory (16-bits addressing)

Parameters:
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values:
- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name: **HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)**

Function description: Write Spare area(s) to NAND memory (16-bits addressing)

Parameters:
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** : number of spare areas to write to block

Return values:
- **HAL:** status

HAL_NAND_Erase_Block

Function name: **HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)**

Function description: NAND memory Block erase.

Parameters:
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure

Return values:
- **HAL:** status

HAL_NAND_Read_Status

Function name: **uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)**

Function description: NAND memory read status.

Parameters:
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values:
- **NAND:** status

HAL_NAND_Address_Inc

Function name **uint32_t HAL_NAND_Address_Inc (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)**

Function description Increment the NAND memory address.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values
- **The:** new status of the increment address operation. It can be:
  - NAND_VALID_ADDRESS: When the new address is valid address
  - NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_ECC_Enable

Function name **HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)**

Function description Enables dynamically NAND ECC feature.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_ECC_Disable

Function name **HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)**

Function description Disables dynamically FSMC_NAND ECC feature.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values
- **HAL:** status

HAL_NAND_GetECC

Function name **HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)**

Function description Disables dynamically NAND ECC feature.

Parameters
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ECCval:** pointer to ECC value
- **Timeout:** maximum timeout to wait

Return values
- **HAL:** status

HAL_NAND_GetState

Function name **HAL_NAND_StateTypeDef HAL_NAND_GetState**

(NAND_HandleTypeDef * hnand)

Function description: return the NAND state

Parameters:
- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values:
- **HAL:** state

27.3 NAND Firmware driver defines

27.3.1 NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:
- Reset NAND handle state.

Parameters:
- __HANDLE__: specifies the NAND handle.

Return value:
- None

28 HAL NOR Generic Driver

28.1 NOR Firmware driver registers structures

28.1.1 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

28.1.2 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***

28.1.3 NOR_HandleTypeDef

Data Fields

- ***FSMC_NORSRAM_TypeDef * Instance***
- ***FSMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FSMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***

Field Documentation

- ***FSMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance***
  Register base address

- ***FSMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FSMC_NORSRAM_InitTypeDef NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef NOR_HandleTypeDef::State***
  NOR device access state

28.2 NOR Firmware driver API description

28.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FSMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()



This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE : NOR memory write data to specified address

28.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Init()***
- ***HAL_NOR_DeInit()***
- ***HAL_NOR_MspInit()***
- ***HAL_NOR_MspDeInit()***
- ***HAL_NOR_MspWait()***

28.2.3 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()***

28.2.4 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()***

28.2.5 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()***

28.2.6 Detailed description of functions

HAL_NOR_Init

Function name	**HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)**
Function description	Perform the NOR memory Initialization sequence.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **Timing:** pointer to NOR control timing structure • **ExtTiming:** pointer to NOR extended mode timing structure
Return values	• **HAL:** status

HAL_NOR_DeInit

Function name	**HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)**
Function description	Perform NOR memory De-Initialization sequence.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values
- **HAL:** status

HAL_NOR_MspInit

Function name **void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)**

Function description NOR MSP Init.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values
- **None:**

HAL_NOR_MspDeInit

Function name **void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)**

Function description NOR MSP DeInit.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values
- **None:**

HAL_NOR_MspWait

Function name **void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)**

Function description NOR MSP Wait fro Ready/Busy signal.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timeout:** Maximum timeout value

Return values
- **None:**

HAL_NOR_Read_ID

Function name **HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)**

Function description Read NOR flash IDs.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_ID:** : pointer to NOR ID structure

Return values
- **HAL:** status

HAL_NOR_ReturnToReadMode

Function name **HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)**

Function description Returns the NOR memory to Read mode.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that

contains the configuration information for NOR module.

Return values
- **HAL:** status

HAL_NOR_Read

Function name **HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)**

Function description Read data from NOR memory.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** pointer to Device address
- **pData:** : pointer to read data

Return values
- **HAL:** status

HAL_NOR_Program

Function name **HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)**

Function description Program data to NOR memory.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** Device address
- **pData:** : pointer to the data to write

Return values
- **HAL:** status

HAL_NOR_ReadBuffer

Function name **HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)**

Function description Reads a block of data from the FSMC NOR memory.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** : number of Half word to read.

Return values
- **HAL:** status

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 FSMC NOR memory.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

- **uwAddress:** NOR memory internal address from which the data
- **pData:** pointer to source data buffer.
- **uwBufferSize:** number of Half words to write.

Return values
- **HAL:** status

Notes
- Some NOR memory need Address aligned to xx bytes (can be aligned to 64 bytes boundary for example).
- The maximum buffer size allowed is NOR memory dependent (can be 64 Bytes max for example).

HAL_NOR_Erase_Block

Function name **HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)**

Function description Erase the specified block of the NOR memory.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** : Block to erase address
- **Address:** Device address

Return values
- **HAL:** status

HAL_NOR_Erase_Chip

Function name **HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)**

Function description Erase the entire NOR chip.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** : Device address

Return values
- **HAL:** status

HAL_NOR_Read_CFI

Function name **HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)**

Function description Read NOR flash CFI IDs.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_CFI:** : pointer to NOR CFI IDs structure

Return values
- **HAL:** status

HAL_NOR_WriteOperation_Enable

Function name **HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)**

Function description Enables dynamically NOR write operation.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values
- **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name **HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)**

Function description Disables dynamically NOR write operation.

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values
- **HAL:** status

HAL_NOR_GetState

Function name **HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)**

Function description return the NOR controller state

Parameters
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values
- **NOR:** controller state

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
- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address
- **Timeout:** NOR progamming Timeout

Return values
- **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

28.3 NOR Firmware driver defines

28.3.1 NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE **Description:**
- Reset NOR handle state.

Parameters:
- __HANDLE__: NOR handle

Return value:

- None

29 HAL PCCARD Generic Driver

29.1 PCCARD Firmware driver registers structures

29.1.1 PCCARD_HandleTypeDef

Data Fields

- ***FSMC_PCCARD_TypeDef * Instance***
- ***FSMC_PCCARD_InitTypeDef Init***
- ***__IO HAL_PCCARD_StateTypeDef State***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***FSMC_PCCARD_TypeDef* PCCARD_HandleTypeDef::Instance***
  Register base address for PCCARD device
- ***FSMC_PCCARD_InitTypeDef PCCARD_HandleTypeDef::Init***
  PCCARD device control configuration parameters
- ***__IO HAL_PCCARD_StateTypeDef PCCARD_HandleTypeDef::State***
  PCCARD device access state
- ***HAL_LockTypeDef PCCARD_HandleTypeDef::Lock***
  PCCARD Lock

29.2 PCCARD Firmware driver API description

29.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control PCCARD/compact flash memories. It uses the FSMC/FSMC layer functions to interface with PCCARD devices. This driver is used for:

- PCCARD/compact flash memory configuration sequence using the function HAL_PCCARD_Init() with control and timing parameters for both common and attribute spaces.
- Read PCCARD/compact flash memory maker and device IDs using the function HAL_PCCARD_Read_ID(). The read information is stored in the CompactFlash_ID structure declared by the function caller.
- Access PCCARD/compact flash memory by read/write operations using the functions HAL_PCCARD_Read_Sector()/HAL_PCCARD_Write_Sector(), to read/write sector.
- Perform PCCARD/compact flash Reset chip operation using the function HAL_PCCARD_Reset().
- Perform PCCARD/compact flash erase sector operation using the function HAL_PCCARD_Erase_Sector().
- Read the PCCARD/compact flash status operation using the function HAL_PCCARD_ReadStatus().
- You can monitor the PCCARD/compact flash device HAL state by calling the function HAL_PCCARD_GetState()



This driver is a set of generic APIs which handle standard PCCARD/compact flash operations. If a PCCARD/compact flash device contains different operations and/or implementations, it should be implemented separately.

29.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

This section contains the following APIs:

- ***HAL_PCCARD_Init()***
- ***HAL_PCCARD_DeInit()***
- ***HAL_PCCARD_MspInit()***
- ***HAL_PCCARD_MspDeInit()***

29.2.3 PCCARD Input Output and memory functions

This section provides functions allowing to use and control the PCCARD memory

This section contains the following APIs:

- ***HAL_PCCARD_Read_ID()***
- ***HAL_PCCARD_Read_Sector()***
- ***HAL_PCCARD_Write_Sector()***
- ***HAL_PCCARD_Erase_Sector()***
- ***HAL_PCCARD_Reset()***
- ***HAL_PCCARD_IRQHandler()***
- ***HAL_PCCARD_ITCallback()***

29.2.4 PCCARD Peripheral State functions

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

This section contains the following APIs:

- ***HAL_PCCARD_GetState()***
- ***HAL_PCCARD_GetStatus()***
- ***HAL_PCCARD_ReadStatus()***

29.2.5 Detailed description of functions

HAL_PCCARD_Init

Function name	**HAL_StatusTypeDef HAL_PCCARD_Init (PCCARD_HandleTypeDef * hpccard, FSMC_NAND_PCC_TimingTypeDef * ComSpaceTiming, FSMC_NAND_PCC_TimingTypeDef * AttSpaceTiming, FSMC_NAND_PCC_TimingTypeDef * IOSpaceTiming)**
Function description	Perform the PCCARD memory Initialization sequence.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module. • **ComSpaceTiming:** Common space timing structure • **AttSpaceTiming:** Attribute space timing structure • **IOSpaceTiming:** IO space timing structure
Return values	• **HAL:** status

HAL_PCCARD_DeInit

Function name **HAL_StatusTypeDef HAL_PCCARD_DeInit (PCCARD_HandleTypeDef * hpccard)**

Function description Perform the PCCARD memory De-initialization sequence.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **HAL:** status

HAL_PCCARD_MspInit

Function name **void HAL_PCCARD_MspInit (PCCARD_HandleTypeDef * hpccard)**

Function description PCCARD MSP Init.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **None:**

HAL_PCCARD_MspDeInit

Function name **void HAL_PCCARD_MspDeInit (PCCARD_HandleTypeDef * hpccard)**

Function description PCCARD MSP DeInit.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **None:**

HAL_PCCARD_Read_ID

Function name **HAL_StatusTypeDef HAL_PCCARD_Read_ID (PCCARD_HandleTypeDef * hpccard, uint8_t CompactFlash_ID, uint8_t * pStatus)**

Function description Read Compact Flash's ID.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **CompactFlash_ID:** Compact flash ID structure.
- **pStatus:** pointer to compact flash status

Return values
- **HAL:** status

HAL_PCCARD_Write_Sector

Function name **HAL_StatusTypeDef HAL_PCCARD_Write_Sector (PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)**

Function description: Write sector to PCCARD memory.

Parameters:
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **pBuffer:** pointer to source write buffer
- **SectorAddress:** Sector address to write
- **pStatus:** pointer to CF status

Return values:
- **HAL:** status

HAL_PCCARD_Read_Sector

Function name: **HAL_StatusTypeDef HAL_PCCARD_Read_Sector (PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)**

Function description: Read sector from PCCARD memory.

Parameters:
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **pBuffer:** pointer to destination read buffer
- **SectorAddress:** Sector address to read
- **pStatus:** pointer to CF status

Return values:
- **HAL:** status

HAL_PCCARD_Erase_Sector

Function name: **HAL_StatusTypeDef HAL_PCCARD_Erase_Sector (PCCARD_HandleTypeDef * hpccard, uint16_t SectorAddress, uint8_t * pStatus)**

Function description: Erase sector from PCCARD memory.

Parameters:
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **SectorAddress:** Sector address to erase
- **pStatus:** pointer to CF status

Return values:
- **HAL:** status

HAL_PCCARD_Reset

Function name: **HAL_StatusTypeDef HAL_PCCARD_Reset (PCCARD_HandleTypeDef * hpccard)**

Function description: Reset the PCCARD memory.

Parameters:
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values:
- **HAL:** status

HAL_PCCARD_IRQHandler

Function name **void HAL_PCCARD_IRQHandler (PCCARD_HandleTypeDef * hpccard)**

Function description This function handles PCCARD device interrupt request.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **HAL:** status

HAL_PCCARD_ITCallback

Function name **void HAL_PCCARD_ITCallback (PCCARD_HandleTypeDef * hpccard)**

Function description PCCARD interrupt feature callback.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **None:**

HAL_PCCARD_GetState

Function name **HAL_PCCARD_StateTypeDef HAL_PCCARD_GetState (PCCARD_HandleTypeDef * hpccard)**

Function description return the PCCARD controller state

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **HAL:** state

HAL_PCCARD_GetStatus

Function name **HAL_PCCARD_StatusTypeDef HAL_PCCARD_GetStatus (PCCARD_HandleTypeDef * hpccard)**

Function description Get the compact flash memory status.

Parameters
- **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.

Return values
- **New:** status of the CF operation. This parameter can be:
  - CompactFlash_TIMEOUT_ERROR: when the previous operation generate a Timeout error
  - CompactFlash_READY: when memory is ready for the next operation

HAL_PCCARD_ReadStatus

Function name **HAL_PCCARD_StatusTypeDef HAL_PCCARD_ReadStatus (PCCARD_HandleTypeDef * hpccard)**

Function description	Reads the Compact Flash memory status using the Read status command.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• **The:** status of the Compact Flash memory. This parameter can be: – CompactFlash_BUSY: when memory is busy – CompactFlash_READY: when memory is ready for the next operation – CompactFlash_ERROR: when the previous operation gererates error

29.3 PCCARD Firmware driver defines

29.3.1 PCCARD

PCCARD Exported Macros

__HAL_PCCARD_RESET_HANDLE_STATE

Description:

- Reset PCCARD handle state.

Parameters:

- __HANDLE__: specifies the PCCARD handle.

Return value:

- None

30 HAL PCD Generic Driver

30.1 PCD Firmware driver registers structures

30.1.1 PCD_HandleTypeDef

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***uint32_t Setup***
- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t PCD_HandleTypeDef::USB_Address***
  USB Address: not used by USB OTG FS
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[15U]***
  IN endpoint parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::OUT_ep[15U]***
  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[12U]***
  Setup packet buffer
- ***void* PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler

30.2 PCD Firmware driver API description

30.2.1 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 the following macro
      – __HAL_RCC_USB_CLK_ENABLE(); For USB Device FS peripheral available on STM32F102xx and STM32F103xx devices

- __HAL_RCC_USB_OTG_FS_CLK_ENABLE(); For USB OTG FS peripheral available on STM32F105xx and STM32F107xx devices

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();

30.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_PCD_Init()***
- ***HAL_PCD_DeInit()***
- ***HAL_PCD_MspInit()***
- ***HAL_PCD_MspDeInit()***

30.2.3 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()***

30.2.4 Peripheral Control functions

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

This section contains the following APIs:

- ***HAL_PCD_DevConnect()***
- ***HAL_PCD_DevDisconnect()***
- ***HAL_PCD_SetAddress()***
- ***HAL_PCD_EP_Open()***
- ***HAL_PCD_EP_Close()***
- ***HAL_PCD_EP_Receive()***
- ***HAL_PCD_EP_GetRxCount()***
- ***HAL_PCD_EP_Transmit()***
- ***HAL_PCD_EP_SetStall()***
- ***HAL_PCD_EP_ClrStall()***

- ***HAL_PCD_EP_Flush()***
- ***HAL_PCD_ActivateRemoteWakeup()***
- ***HAL_PCD_DeActivateRemoteWakeup()***

30.2.5 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()***

30.2.6 Detailed description of functions

HAL_PCD_Init

Function name	**HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)**
Function description	Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and create the associated handle.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DeInit

Function name	**HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)**
Function description	DeInitializes the PCD peripheral.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_MspInit

Function name	**void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)**
Function description	Initializes the PCD MSP.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_MspDeInit

Function name	**void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)**
Function description	DeInitializes PCD MSP.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_Start

Function name	**HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)**

Function description	Start The USB Device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_Stop

Function name	**HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)**
Function description	Stop The USB Device.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_IRQHandler

Function name	**void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)**
Function description	This function handles PCD interrupt request.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DataOutStageCallback

Function name	**void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Data out stage callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None:**

HAL_PCD_DataInStageCallback

Function name	**void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Data IN stage callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None:**

HAL_PCD_SetupStageCallback

Function name	**void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)**
Function description	Setup stage callback.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_SOFCallback

Function name	**void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)**
Function description	USB Start Of Frame callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_ResetCallback

Function name	**void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)**
Function description	USB Reset callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_SuspendCallback

Function name	**void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)**
Function description	Suspend event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_ResumeCallback

Function name	**void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)**
Function description	Resume event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name	**void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Incomplete ISO OUT callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• **None:**

HAL_PCD_ISOINIncompleteCallback

Function name	**void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function description	Incomplete ISO IN callbacks.
Parameters	• **hpcd:** PCD handle

- **epnum:** endpoint number

Return values
- **None:**

HAL_PCD_ConnectCallback

Function name **void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)**

Function description Connection event callbacks.

Parameters
- **hpcd:** PCD handle

Return values
- **None:**

HAL_PCD_DisconnectCallback

Function name **void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)**

Function description Disconnection event callbacks.

Parameters
- **hpcd:** PCD handle

Return values
- **None:**

HAL_PCD_DevConnect

Function name **HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)**

Function description Connect the USB device.

Parameters
- **hpcd:** PCD handle

Return values
- **HAL:** status

HAL_PCD_DevDisconnect

Function name **HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)**

Function description Disconnect the USB device.

Parameters
- **hpcd:** PCD handle

Return values
- **HAL:** status

HAL_PCD_SetAddress

Function name **HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)**

Function description Set the USB Device address.

Parameters
- **hpcd:** PCD handle
- **address:** new device address

Return values
- **HAL:** status

HAL_PCD_EP_Open

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)**
Function description	Open and configure an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **ep_mps:** endpoint max packet size • **ep_type:** endpoint type
Return values	• **HAL:** status

HAL_PCD_EP_Close

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Deactivate an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_Receive

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)**
Function description	Receive an amount of data.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **pBuf:** pointer to the reception buffer • **len:** amount of data to be received
Return values	• **HAL:** status

HAL_PCD_EP_Transmit

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)**
Function description	Send an amount of data.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address • **pBuf:** pointer to the transmission buffer • **len:** amount of data to be sent
Return values	• **HAL:** status

HAL_PCD_EP_GetRxCount

Function name	**uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Get Received Data Size.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **Data:** Size

HAL_PCD_EP_SetStall

Function name	**HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Set a STALL condition over an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_ClrStall

Function name	**HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Clear a STALL condition over in an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_EP_Flush

Function name	**HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**
Function description	Flush an endpoint.
Parameters	• **hpcd:** PCD handle • **ep_addr:** endpoint address
Return values	• **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name	**HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function description	HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_DeActivateRemoteWakeup

Function name	**HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function description	HAL_PCD_DeActivateRemoteWakeup : de-active remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** status

HAL_PCD_GetState

Function name	**PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)**
Function description	Return the PCD state.
Parameters	• **hpcd:** PCD handle
Return values	• **HAL:** state

30.3 PCD Firmware driver defines

30.3.1 PCD

PCD ENDP

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD Endpoint Kind

PCD_SNG_BUF

PCD_DBL_BUF

PCD EP0 MPS

PCD_EP0MPS_64

PCD_EP0MPS_32

PCD_EP0MPS_16

PCD_EP0MPS_08

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_WAKEUP_EXTI_GET_FLAG

__HAL_USB_WAKEUP_EXTI_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_RISING_EDGE

__HAL_USB_WAKEUP_EXTI_ENABLE_FALLING_EDGE

__HAL_USB_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

PCD Instance definition

IS_PCD_ALL_INSTANCE

PCD PHY Module

PCD_PHY_EMBEDDED

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_HIGH_IN_FULL

PCD_SPEED_FULL

Turnaround Timeout Value

USBD_FS_TRDT_VALUE

31 HAL PCD Extension Driver

31.1 PCDEx Firmware driver API description

31.1.1 Extended Peripheral Control functions

This section provides functions allowing to:

- Update FIFO (USB_OTG_FS)
- Update PMA configuration (USB)

This section contains the following APIs:

- ***HAL_PCDEx_PMAConfig()***

31.1.2 Detailed description of functions

HAL_PCDEx_PMAConfig

Function name
HAL_StatusTypeDef HAL_PCDEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaadress)

Function description
Configure PMA for EP.

Parameters
- **hpcd:** : Device instance
- **ep_addr:** endpoint address
- **ep_kind:** endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used
- **pmaadress:** EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.

Return values
- **HAL:** status

HAL_PCDEx_SetConnectionState

Function name
void HAL_PCDEx_SetConnectionState (PCD_HandleTypeDef * hpcd, uint8_t state)

Function description
Software Device Connection, this function is not required by USB OTG FS peripheral, it is used only by USB Device FS peripheral.

Parameters
- **hpcd:** PCD handle
- **state:** connection state (0 : disconnected / 1: connected)

Return values
- **None:**

32 HAL PWR Generic Driver

32.1 PWR Firmware driver registers structures

32.1.1 PWR_PVDTypeDef

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***
  PVDLevel: Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_detection_level***
- ***uint32_t PWR_PVDTypeDef::Mode***
  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***

32.2 PWR Firmware driver API description

32.2.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers) 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()***

32.2.2 Peripheral Control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through __HAL_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There is one WakeUp pin: WakeUp Pin 1 on PA.00.

Low Power modes configuration

The device features 3 low-power modes:

- Sleep mode: CPU clock off, all peripherals including Cortex-M3 core peripherals like NVIC, SysTick, etc. are kept running
- Stop mode: All clocks are stopped
- Standby mode: 1.8V domain powered off

Sleep mode

- Entry: The Sleep mode is entered by using the HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFx) functions with
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- Exit:
  - WFI entry mode, Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.
  - WFE entry mode, Any wakeup event can wake up the device from Sleep mode.
    - Any peripheral interrupt w/o NVIC configuration & SEVONPEND bit set in the Cortex (HAL_PWR_EnableSEVOnPend)
    - Any EXTI Line (Internal or External) configured in Event mode

Stop mode

The Stop mode is based on the Cortex-M3 deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the 1.8 V domain are stopped, the PLL, the HSI and the HSE RC oscillators are disabled. SRAM and register contents are preserved. In Stop mode, all I/O pins keep the same state as in Run mode.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode(PWR_REGULATOR_VALUE, PWR_SLEEPENTRY_WFx) function with:
  - PWR_REGULATOR_VALUE= PWR_MAINREGULATOR_ON: Main regulator ON.
  - PWR_REGULATOR_VALUE= PWR_LOWPOWERREGULATOR_ON: Low Power regulator ON.
  - PWR_SLEEPENTRY_WFx= PWR_SLEEPENTRY_WFI: enter STOP mode with WFI instruction
  - PWR_SLEEPENTRY_WFx= PWR_SLEEPENTRY_WFE: enter STOP mode with WFE instruction
- Exit:
  - WFI entry mode, Any EXTI Line (Internal or External) configured in Interrupt mode with NVIC configured
  - WFE entry mode, Any EXTI Line (Internal or External) configured in Event mode.

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M3 deepsleep mode, with the voltage regulator disabled. The 1.8 V 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 registers in the Backup domain and Standby circuitry

- Entry:

  - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:
  - WKUP pin rising edge, RTC alarm event rising edge, external Reset in NRSTpin, IWDG Reset

Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm 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.

PWR Workarounds linked to Silicon Limitation

Below the list of all silicon limitations known on STM32F1xx prouct.

1. Workarounds Implemented inside PWR HAL Driver
   a. Debugging Stop mode with WFE entry - overloaded the WFE by an internal 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_EnableSleepOnExit()***
- ***HAL_PWR_DisableSleepOnExit()***
- ***HAL_PWR_EnableSEVOnPend()***
- ***HAL_PWR_DisableSEVOnPend()***
- ***HAL_PWR_PVD_IRQHandler()***
- ***HAL_PWR_PVDCallback()***

32.2.3 Detailed description of functions

HAL_PWR_DeInit

Function name	**void HAL_PWR_DeInit (void)**
Function description	Deinitializes the PWR peripheral registers to their default reset values.
Return values	• **None:**

HAL_PWR_EnableBkUpAccess

Function name	**void HAL_PWR_EnableBkUpAccess (void)**
Function description	Enables access to the backup domain (RTC registers, RTC backup data registers).

Return values
- **None:**

Notes
- If the HSE divided by 128 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_DisableBkUpAccess

Function name **void HAL_PWR_DisableBkUpAccess (void)**

Function description Disables access to the backup domain (RTC registers, RTC backup data registers).

Return values
- **None:**

Notes
- If the HSE divided by 128 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_ConfigPVD

Function name **void HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)**

Function description Configures the voltage threshold detected by the Power Voltage Detector(PVD).

Parameters
- **sConfigPVD:** pointer to an PWR_PVDTypeDef structure that contains the configuration information for the PVD.

Return values
- **None:**

Notes
- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

HAL_PWR_EnablePVD

Function name **void HAL_PWR_EnablePVD (void)**

Function description Enables the Power Voltage Detector(PVD).

Return values
- **None:**

HAL_PWR_DisablePVD

Function name **void HAL_PWR_DisablePVD (void)**

Function description Disables the Power Voltage Detector(PVD).

Return values
- **None:**

HAL_PWR_EnableWakeUpPin

Function name **void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)**

Function description Enables the WakeUp PINx functionality.

Parameters
- **WakeUpPinx:** Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values:
  – PWR_WAKEUP_PIN1

Return values
- **None:**

HAL_PWR_DisableWakeUpPin

Function name **void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)**

Function description Disables the WakeUp PINx functionality.

Parameters
- **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
  – PWR_WAKEUP_PIN1

Return values
- **None:**

HAL_PWR_EnterSTOPMode

Function name **void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)**

Function description Enters Stop mode.

Parameters
- **Regulator:** Specifies the regulator state in Stop mode. This parameter can be one of the following values:
  – PWR_MAINREGULATOR_ON: Stop mode with regulator ON
  – PWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON
- **STOPEntry:** Specifies if Stop mode in entered with WFI or WFE instruction. This parameter can be one of the following values:
  – PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction
  – PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction

Return values
- **None:**

Notes
- In Stop mode, all I/O pins keep the same state as in Run mode.
- When exiting Stop mode by using an interrupt or a wakeup event, 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.

HAL_PWR_EnterSLEEPMode

Function name **void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)**

Function description Enters Sleep mode.

Parameters
- **Regulator:** Regulator state as no effect in SLEEP mode - allows to support portability from legacy software
- **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source. This parameter can be one of the following values:
  – PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI

instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction

Return values
- **None:**

Notes
- In Sleep mode, all I/O pins keep the same state as in Run mode.

HAL_PWR_EnterSTANDBYMode

Function name **void HAL_PWR_EnterSTANDBYMode (void)**

Function description Enters Standby mode.

Return values
- **None:**

Notes
- In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)TAMPER pin if configured for tamper or calibration out.WKUP pin (PA0) if enabled.

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
- **None:**

Notes
- 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.

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
- **None:**

Notes
- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name **void HAL_PWR_EnableSEVOnPend (void)**

Function description Enables CORTEX M3 SEVONPEND bit.

Return values
- **None:**

Notes
- Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_DisableSEVOnPend

Function name	**void HAL_PWR_DisableSEVOnPend (void)**
Function description	Disables CORTEX M3 SEVONPEND bit.
Return values	• **None:**
Notes	• Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_PVD_IRQHandler

Function name	**void HAL_PWR_PVD_IRQHandler (void)**
Function description	This function handles the PWR PVD interrupt request.
Return values	• **None:**
Notes	• This API should be called under the PVD_IRQHandler().

HAL_PWR_PVDCallback

Function name	**void HAL_PWR_PVDCallback (void)**
Function description	PWR PVD interrupt callback.
Return values	• **None:**

32.3 PWR Firmware driver defines

32.3.1 PWR

PWR CR Register alias address

LPSDSR_BIT_NUMBER

CR_LPSDSR_BB

DBP_BIT_NUMBER

CR_DBP_BB

PVDE_BIT_NUMBER

CR_PVDE_BB

PWR CSR Register alias address

CSR_EWUP_BB

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check PWR flag is set or not.

Parameters:

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

– PWR_FLAG_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
– PWR_FLAG_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
– PWR_FLAG_PVDO: PVD Output. This flag is valid only if PVD is enabled by the HAL_PWR_EnablePVD() function. The PVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.

Return value:

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

__HAL_PWR_CLEAR_FLAG

Description:

- Clear the PWR's pending flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  – PWR_FLAG_WU: Wake Up flag
  – PWR_FLAG_SB: StandBy flag

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable interrupt on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable interrupt on PVD Exti Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_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_FALLING_EDGE

Description:

- PVD EXTI line configuration: set falling edge 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_EDGE

Description:

- PVD EXTI line configuration: set rising edge 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_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:** • Check whether the specified PVD EXTI interrupt flag is set or not. **Return value:** • EXTI: PVD Line Status.
__HAL_PWR_PVD_EXTI_CLEAR_FLAG	**Description:** • Clear the PVD EXTI flag. **Return value:** • None.
__HAL_PWR_PVD_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.

PWR Flag

PWR_FLAG_WU

PWR_FLAG_SB

PWR_FLAG_PVDO

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 Mode	
PWR_PVD_MODE_NORMAL	basic mode is used
PWR_PVD_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection
PWR PVD Mode Mask	
PVD_MODE_IT	
PVD_MODE_EVT	
PVD_RISING_EDGE	
PVD_FALLING_EDGE	
PWR Register alias address	
PWR_OFFSET	
PWR_CR_OFFSET	
PWR_CSR_OFFSET	
PWR_CR_OFFSET_BB	
PWR_CSR_OFFSET_BB	
PWR Regulator state in SLEEP/STOP mode	
PWR_MAINREGULATOR_ON	
PWR_LOWPOWERREGULATOR_ON	
PWR SLEEP mode entry	
PWR_SLEEPENTRY_WFI	
PWR_SLEEPENTRY_WFE	
PWR STOP mode entry	
PWR_STOPENTRY_WFI	
PWR_STOPENTRY_WFE	
PWR WakeUp Pins	
PWR_WAKEUP_PIN1	

33 HAL RCC Generic Driver

33.1 RCC Firmware driver registers structures

33.1.1 RCC_PLLInitTypeDef

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLMUL***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  PLLState: The new state of the PLL. This parameter can be a value of ***RCC_PLL_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
  PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLMUL***
  PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of ***RCCEx_PLL_Multiplication_Factor***

33.1.2 RCC_ClkInitTypeDef

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source (SYSCLKS) used as system clock. This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***

33.2 RCC Firmware driver API description

33.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 8MHz) with Flash 0 wait state, Flash prefetch buffer is enabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) buses; all peripherals mapped on these buses 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 buses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (I2S, RTC, ADC, USB OTG FS)

33.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
  - AHB & APB peripherals, 1 dummy read is necessary

Workarounds:

1. For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each __HAL_RCC_PPP_CLK_ENABLE() macro.

33.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/external oscillators (HSE, HSI, LSE, LSI, PLL, CSS and MCO) and the System buses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 8 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. LSI (low-speed internal), ~40 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 24 MHz (STM32F100xx) or 4 to 16 MHz (STM32F101x/STM32F102x/STM32F103x) or 3 to 25 MHz (STM32F105x/STM32F107x) crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
4. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
5. PLL (clocked by HSI or HSE), featuring different output clocks:
   - The first output is used to generate the high speed system clock (up to 72 MHz for STM32F10xxx or up to 24 MHz for STM32F100xx)
   - The second output is used to generate the clock for the USB OTG FS (48 MHz)

6. CSS (Clock security system), once enable using the macro __HAL_RCC_CSS_ENABLE() and if a HSE clock failure occurs(HSE used directly or through PLL as System clock source), the System clocks automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
7. MCO1 (microcontroller clock output), used to output SYSCLK, HSI, HSE or PLL clock (divided by 2) on PA8 pin + PLL2CLK, PLL3CLK/2, PLL3CLK and XTI for STM32F105x/STM32F107x

System, AHB and APB buses 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 buses. You can use "@ref HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: RTC: RTC clock can be derived either from the LSI, LSE or HSE clock divided by 128. USB OTG FS and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly. This clock is derived of the main PLL through PLL Multiplier.I2S interface on STM32F105x/STM32F107x can be derived from PLL3CLKIWDG clock which is always the LSI clock.
2. For STM32F10xxx, the maximum frequency of the SYSCLK and HCLK/PCLK2 is 72 MHz, PCLK1 36 MHz. For STM32F100xx, the maximum frequency of the SYSCLK and HCLK/PCLK1/PCLK2 is 24 MHz. Depending on the SYSCLK frequency, the flash latency should be adapted accordingly.

This section contains the following APIs:

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

33.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- ***HAL_RCC_MCOConfig()***
- ***HAL_RCC_EnableCSS()***
- ***HAL_RCC_DisableCSS()***
- ***HAL_RCC_GetSysClockFreq()***
- ***HAL_RCC_GetHCLKFreq()***
- ***HAL_RCC_GetPCLK1Freq()***
- ***HAL_RCC_GetPCLK2Freq()***
- ***HAL_RCC_GetOscConfig()***
- ***HAL_RCC_GetClockConfig()***
- ***HAL_RCC_NMI_IRQHandler()***
- ***HAL_RCC_CSSCallback()***

33.2.5 Detailed description of functions

HAL_RCC_DeInit

Function name **void HAL_RCC_DeInit (void)**

Function description Resets the RCC clock configuration to the default reset state.

Return values
- **None:**

Notes
- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS and MCO1 OFFAll interrupts disabled
- This function does not modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name **HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)**

Function description Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters
- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values
- **HAL:** status

Notes
- The PLL is not disabled when used as system clock.
- The PLL is not disabled when USB OTG FS clock is enabled (specific to devices with USB FS)
- Transitions 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.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

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 buses clocks according to the specified parameters in the RCC_ClkInitStruct.

Parameters
- **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.
- **FLatency:** FLASH Latency The value of this parameter depend on device used within the same series

Return values
- **HAL:** status

Notes
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function
- The HSI is used (enabled by hardware) as system clock source after start-up 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 start-up 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.

HAL_RCC_MCOConfig

Function name **void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)**

Function description Selects the clock source to output on MCO pin.

Parameters
- **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_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK No clock selected as MCO clock
  - RCC_MCO1SOURCE_SYSCLK System clock selected as MCO clock
  - RCC_MCO1SOURCE_HSI HSI selected as MCO clock
  - RCC_MCO1SOURCE_HSE HSE selected as MCO clock
- **RCC_MCODiv:** specifies the MCO DIV. This parameter can be one of the following values:
  - RCC_MCODIV_1 no division applied to MCO clock

Return values
- **None:**

Notes
- MCO pin should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name **void HAL_RCC_EnableCSS (void)**

Function description Enables the Clock Security System.

Return values
- **None:**

Notes
- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.

HAL_RCC_DisableCSS

Function name **void HAL_RCC_DisableCSS (void)**

Function description Disables the Clock Security System.

Return values
- **None:**

HAL_RCC_GetSysClockFreq

Function name	**uint32_t HAL_RCC_GetSysClockFreq (void)**
Function description	Returns the SYSCLK frequency.
Return values	• **SYSCLK:** frequency
Notes	• The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source: • If SYSCLK source is HSI, function returns values based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**) • If SYSCLK source is PLL, function returns a value based on HSE_VALUE divided by PREDIV factor(**) or HSI_VALUE(*) multiplied by the PLL factor. • (*) HSI_VALUE is a constant defined in stm32f1xx_hal_conf.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature. • (**) HSE_VALUE is a constant defined in stm32f1xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result. • The result of this function could be not correct when using fractional value for HSE crystal. • This function can be used by the user application to compute the baud-rate 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.

HAL_RCC_GetHCLKFreq

Function name	**uint32_t HAL_RCC_GetHCLKFreq (void)**
Function description	Returns the HCLK frequency.
Return values	• **HCLK:** frequency
Notes	• Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect. • The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

HAL_RCC_GetPCLK1Freq

Function name	**uint32_t HAL_RCC_GetPCLK1Freq (void)**
Function description	Returns the PCLK1 frequency.
Return values	• **PCLK1:** frequency
Notes	• Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration

based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name	**uint32_t HAL_RCC_GetPCLK2Freq (void)**
Function description	Returns the PCLK2 frequency.
Return values	• **PCLK2:** frequency
Notes	• Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name	**void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)**
Function description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	• **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	• **None:**

HAL_RCC_GetClockConfig

Function name	**void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)**
Function description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	• **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that contains the current clock configuration. • **pFLatency:** Pointer on the Flash Latency.
Return values	• **None:**

HAL_RCC_NMI_IRQHandler

Function name	**void HAL_RCC_NMI_IRQHandler (void)**
Function description	This function handles the RCC CSS interrupt request.
Return values	• **None:**
Notes	• This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name	**void HAL_RCC_CSSCallback (void)**
Function description	RCC Clock Security System interrupt callback.
Return values	• **none:**

33.3 RCC Firmware driver defines

33.3.1 RCC

AHB Clock Source

RCC_SYSCLK_DIV1	SYSCLK not divided
RCC_SYSCLK_DIV2	SYSCLK divided by 2
RCC_SYSCLK_DIV4	SYSCLK divided by 4
RCC_SYSCLK_DIV8	SYSCLK divided by 8
RCC_SYSCLK_DIV16	SYSCLK divided by 16
RCC_SYSCLK_DIV64	SYSCLK divided by 64
RCC_SYSCLK_DIV128	SYSCLK divided by 128
RCC_SYSCLK_DIV256	SYSCLK divided by 256
RCC_SYSCLK_DIV512	SYSCLK divided by 512

AHB Peripheral Clock Enable Disable Status

__HAL_RCC_DMA1_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_SRAM_IS_CLK_ENABLED

__HAL_RCC_SRAM_IS_CLK_DISABLED

__HAL_RCC_FLITF_IS_CLK_ENABLED

__HAL_RCC_FLITF_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

Alias define maintained for legacy

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

APB1 APB2 Clock Source

RCC_HCLK_DIV1	HCLK not divided
RCC_HCLK_DIV2	HCLK divided by 2
RCC_HCLK_DIV4	HCLK divided by 4
RCC_HCLK_DIV8	HCLK divided by 8
RCC_HCLK_DIV16	HCLK divided by 16

APB1 Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_BKP_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_WWDG_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_BKP_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

APB1 Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_WWDG_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_BKP_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_WWDG_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_BKP_RELEASE_RESET

__HAL_RCC_PWR_RELEASE_RESET

APB1 Peripheral Clock Enable Disable Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_BKP_IS_CLK_ENABLED

__HAL_RCC_BKP_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

APB2 Clock Enable Disable

__HAL_RCC_AFIO_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_ADC1_CLK_ENABLE

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_AFIO_CLK_DISABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_ADC1_CLK_DISABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

APB2 Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_AFIO_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_ADC1_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_AFIO_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_ADC1_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

APB2 Peripheral Clock Enable Disable Status

__HAL_RCC_AFIO_IS_CLK_ENABLED

__HAL_RCC_AFIO_IS_CLK_DISABLED

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_ADC1_IS_CLK_ENABLED

__HAL_RCC_ADC1_IS_CLK_DISABLED

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

BitAddress AliasRegion

RCC_CR_OFFSET_BB

RCC_CFGR_OFFSET_BB

RCC_CIR_OFFSET_BB

RCC_BDCR_OFFSET_BB

RCC_CSR_OFFSET_BB

RCC_HSION_BIT_NUMBER

RCC_CR_HSION_BB

RCC_HSEON_BIT_NUMBER

RCC_CR_HSEON_BB

RCC_CSSON_BIT_NUMBER

RCC_CR_CSSON_BB

RCC_PLLON_BIT_NUMBER

RCC_CR_PLLON_BB

RCC_LSION_BIT_NUMBER

RCC_CSR_LSION_BB

RCC_RMVF_BIT_NUMBER

RCC_CSR_RMVF_BB

RCC_LSEON_BIT_NUMBER

RCC_BDCR_LSEON_BB

RCC_LSEBYP_BIT_NUMBER

RCC_BDCR_LSEBYP_BB

RCC_RTCEN_BIT_NUMBER

RCC_BDCR_RTCEN_BB

RCC_BDRST_BIT_NUMBER

RCC_BDCR_BDRST_BB

Flags

RCC_FLAG_HSIRDY	Internal High Speed clock ready flag
RCC_FLAG_HSERDY	External High Speed clock ready flag
RCC_FLAG_PLLRDY	PLL clock ready flag
RCC_FLAG_LSIRDY	Internal Low Speed oscillator Ready
RCC_FLAG_PINRST	PIN reset flag
RCC_FLAG_PORRST	POR/PDR reset flag
RCC_FLAG_SFTRST	Software Reset flag
RCC_FLAG_IWDGRST	Independent Watchdog reset flag
RCC_FLAG_WWDGRST	Window watchdog reset flag
RCC_FLAG_LPWRRST	Low-Power reset flag
RCC_FLAG_LSERDY	External Low Speed oscillator Ready

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt.

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

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt.

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

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's 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_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_CSS Clock Security System interrupt

Return value:

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

__HAL_RCC_CLEAR_RESET_FLAGS

The reset flags are RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST

__HAL_RCC_GET_FLAG

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_LSERDY LSE oscillator clock ready.
  - RCC_FLAG_LSIRDY LSI oscillator clock ready.
  - 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).

Get Clock source

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system

clock source. This parameter can be one of the following values:
- RCC_SYSCLKSOURCE_HSI HSI oscillator is used as system clock source.
- RCC_SYSCLKSOURCE_HSE HSE oscillator is used as system clock source.
- RCC_SYSCLKSOURCE_PLLCLK PLL output is used as system clock source.

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_STATUS_HSI HSI used as system clock
  - RCC_SYSCLKSOURCE_STATUS_HSE HSE used as system clock
  - RCC_SYSCLKSOURCE_STATUS_PLLCLK PLL used as system clock

HSE Config

RCC_HSE_OFF	HSE clock deactivation
RCC_HSE_ON	HSE clock activation
RCC_HSE_BYPASS	External clock source for HSE clock

HSE Configuration

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC_HSE_OFF turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC_HSE_ON turn ON the HSE oscillator
  - RCC_HSE_BYPASS HSE oscillator bypassed with external clock

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the

PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

HSI Config

RCC_HSI_OFF	HSI clock deactivation
RCC_HSI_ON	HSI clock activation
RCC_HSICALIBRATION_DEFAULT	

HSI Configuration

__HAL_RCC_HSI_ENABLE

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. 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. (default is RCC_HSICALIBRATION_DEFAULT). 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.

Interrupts

RCC_IT_LSIRDY	LSI Ready Interrupt flag
RCC_IT_LSERDY	LSE Ready Interrupt flag
RCC_IT_HSIRDY	HSI Ready Interrupt flag
RCC_IT_HSERDY	HSE Ready Interrupt flag
RCC_IT_PLLRDY	PLL Ready Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag

LSE Config

RCC_LSE_OFF	LSE clock deactivation
RCC_LSE_ON	LSE clock activation
RCC_LSE_BYPASS	External clock source for LSE clock

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:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

LSI Config

RCC_LSI_OFF	LSI clock deactivation
RCC_LSI_ON	LSI clock activation

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.

__HAL_RCC_LSI_DISABLE **Notes:**

- 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.

MCO Clock Prescaler

RCC_MCODIV_1

MCO Index

RCC_MCO1

RCC_MCO MCO1 to be compliant with other families with 2 MCOs

Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE

RCC_OSCILLATORTYPE_HSI

RCC_OSCILLATORTYPE_LSE

RCC_OSCILLATORTYPE_LSI

Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE

__HAL_RCC_SRAM_CLK_ENABLE

__HAL_RCC_FLITF_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_SRAM_CLK_DISABLE

__HAL_RCC_FLITF_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

PLL Clock Source

RCC_PLLSOURCE_HSI_DIV2 HSI clock divided by 2 selected as PLL entry clock source

RCC_PLLSOURCE_HSE HSE clock selected as PLL entry clock source

PLL Config

RCC_PLL_NONE PLL is not configured

RCC_PLL_OFF PLL deactivation

RCC_PLL_ON PLL activation

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 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

Notes:

- The main PLL can not be disabled if it is used as system clock source

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configure the main PLL clock source and multiplication factors.

Parameters:

- __RCC_PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI_DIV2 HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
- __PLLMUL__: specifies the multiplication factor for PLL VCO output clock This parameter can be one of the following values:
  - RCC_PLL_MUL4 PLLVCO = PLL clock entry x 4
  - RCC_PLL_MUL6 PLLVCO = PLL clock entry x 6
  - RCC_PLL_MUL2 PLLVCO = PLL clock entry x 2
  - RCC_PLL_MUL3 PLLVCO = PLL clock entry x 3
  - RCC_PLL_MUL10 PLLVCO = PLL clock entry x 10
  - RCC_PLL_MUL11 PLLVCO = PLL clock entry x 11
  - RCC_PLL_MUL12 PLLVCO = PLL clock entry x 12
  - RCC_PLL_MUL13 PLLVCO = PLL clock entry x 13
  - RCC_PLL_MUL14 PLLVCO = PLL clock entry x 14
  - RCC_PLL_MUL15 PLLVCO = PLL clock entry x 15
  - RCC_PLL_MUL16 PLLVCO = PLL clock entry x 16

- RCC_PLL_MUL8 PLLVCO = PLL clock entry x 8
- RCC_PLL_MUL9 PLLVCO = PLL clock entry x 9

Notes:

- This function must be used only when the main PLL is disabled.

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Get oscillator clock selected as PLL input clock.

Return value:

- The: clock source used for PLL entry. The returned value can be one of the following:
  - RCC_PLLSOURCE_HSI_DIV2 HSI oscillator clock selected as PLL input clock
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL input clock

Register offsets

RCC_OFFSET

RCC_CR_OFFSET

RCC_CFGR_OFFSET

RCC_CIR_OFFSET

RCC_BDCR_OFFSET

RCC_CSR_OFFSET

RCC RTC Clock Configuration

__HAL_RCC_RTC_CONFIG

Description:

- Macro to configure the RTC clock (RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV128 HSE divided by 128 selected as RTC clock

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_FORCE() 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_GET_RTC_SOURCE

Description:

- Macro to get the RTC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV128 HSE divided by 128 selected as RTC clock

__HAL_RCC_RTC_ENABLE

Notes:

- These macros must be used only after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

Notes:

- These macros must be used only after the RTC clock source was selected.

__HAL_RCC_BACKUPRESET_FORCE

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_BDCR register.

__HAL_RCC_BACKUPRESET_RELEASE

RTC Clock Source

RCC_RTCCLKSOURCE_NO_CLK	No clock
RCC_RTCCLKSOURCE_LSE	LSE oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_LSI	LSI oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV128	HSE oscillator clock divided by 128 used as RTC clock

System Clock Source

RCC_SYSCLKSOURCE_HSI	HSI selected as system clock
RCC_SYSCLKSOURCE_HSE	HSE selected as system clock
RCC_SYSCLKSOURCE_PLLCLK	PLL selected as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI	HSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSE	HSE used as system clock
RCC_SYSCLKSOURCE_STATUS_PLLCLK	PLL used as system clock

System Clock Type

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure
RCC_CLOCKTYPE_PCLK2	PCLK2 to configure

RCC Timeout

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

CLOCKSWITCH_TIMEOUT_VALUE

HSE_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

LSI_TIMEOUT_VALUE

PLL_TIMEOUT_VALUE

34 HAL RCC Extension Driver

34.1 RCCEx Firmware driver registers structures

34.1.1 RCC_OscInitTypeDef

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t HSEPredivValue***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***RCC_PLLInitTypeDef PLL***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***
- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSEPredivValue***
  The Prediv1 factor value (named PREDIV1 or PLLXTPRE in RM) This parameter can be a value of ***RCCEx_Prediv1_Factor***
- ***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 HSI 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

34.1.2 RCC_PeriphCLKInitTypeDef

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t AdcClockSelection***
- ***uint32_t I2s2ClockSelection***
- ***uint32_t I2s3ClockSelection***
- ***uint32_t UsbClockSelection***

Field Documentation

- ***uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***

- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  specifies the RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection***
  ADC clock source This parameter can be a value of ***RCCEx_ADC_Prescaler***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2s2ClockSelection***
  I2S2 clock source This parameter can be a value of ***RCCEx_I2S2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2s3ClockSelection***
  I2S3 clock source This parameter can be a value of ***RCCEx_I2S3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection***
  USB clock source This parameter can be a value of ***RCCEx_USB_Prescaler***

34.2 RCCEx Firmware driver API description

34.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.



Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***

34.2.2 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name	**HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function description	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(RTC clock).
Return values	• **HAL:** status
Notes	• Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values. • In case of STM32F105xC or STM32F107xC devices, PLLI2S will be enabled if requested on one of 2 I2S interfaces. When PLLI2S is enabled, you need to call HAL_RCCEx_DisablePLLI2S to manually disable it.

HAL_RCCEx_GetPeriphCLKConfig

Function name	**void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function description	Get the PeriphClkInit according to the internal RCC configuration registers.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(RTC, I2S, ADC clocks).
Return values	• **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name	**uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)**
Function description	Returns the peripheral clock frequency.
Parameters	• **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values: – RCC_PERIPHCLK_RTC RTC peripheral clock – RCC_PERIPHCLK_ADC ADC peripheral clock – RCC_PERIPHCLK_I2S2 I2S2 peripheral clock – RCC_PERIPHCLK_I2S3 I2S3 peripheral clock – RCC_PERIPHCLK_I2S3 I2S3 peripheral clock – RCC_PERIPHCLK_I2S2 I2S2 peripheral clock
Return values	• **Frequency:** in Hz (0: means that no available frequency for the peripheral)
Notes	• Returns 0 if peripheral clock is unknown

34.3 RCCEx Firmware driver defines

34.3.1 RCCEx

ADC Prescaler

RCC_ADCPCLK2_DIV2

RCC_ADCPCLK2_DIV4

RCC_ADCPCLK2_DIV6

RCC_ADCPCLK2_DIV8

AHB1 Peripheral Clock Enable Disable Status

__HAL_RCC_DMA2_IS_CLK_ENABLED

__HAL_RCC_DMA2_IS_CLK_DISABLED

__HAL_RCC_FSMC_IS_CLK_ENABLED

__HAL_RCC_FSMC_IS_CLK_DISABLED

__HAL_RCC_SDIO_IS_CLK_ENABLED

__HAL_RCC_SDIO_IS_CLK_DISABLED

APB1 Clock Enable Disable

__HAL_RCC_CAN1_CLK_ENABLE

__HAL_RCC_CAN1_CLK_DISABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_USB_CLK_ENABLE

__HAL_RCC_USB_CLK_DISABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_DAC_CLK_ENABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_DAC_CLK_DISABLE

__HAL_RCC_TIM12_CLK_ENABLE

__HAL_RCC_TIM13_CLK_ENABLE

__HAL_RCC_TIM14_CLK_ENABLE

__HAL_RCC_TIM12_CLK_DISABLE

__HAL_RCC_TIM13_CLK_DISABLE

__HAL_RCC_TIM14_CLK_DISABLE

APB1 Force Release Reset

__HAL_RCC_CAN1_FORCE_RESET

__HAL_RCC_CAN1_RELEASE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_USB_FORCE_RESET

__HAL_RCC_USB_RELEASE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_DAC_FORCE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_DAC_RELEASE_RESET

__HAL_RCC_TIM12_FORCE_RESET

__HAL_RCC_TIM13_FORCE_RESET

__HAL_RCC_TIM14_FORCE_RESET

__HAL_RCC_TIM12_RELEASE_RESET

__HAL_RCC_TIM13_RELEASE_RESET

__HAL_RCC_TIM14_RELEASE_RESET

APB1 Peripheral Clock Enable Disable Status

__HAL_RCC_CAN1_IS_CLK_ENABLED

__HAL_RCC_CAN1_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_USB_IS_CLK_ENABLED

__HAL_RCC_USB_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_DAC_IS_CLK_ENABLED

__HAL_RCC_DAC_IS_CLK_DISABLED

__HAL_RCC_TIM13_IS_CLK_ENABLED

__HAL_RCC_TIM13_IS_CLK_DISABLED

__HAL_RCC_TIM14_IS_CLK_ENABLED

__HAL_RCC_TIM14_IS_CLK_DISABLED

APB2 Clock Enable Disable

__HAL_RCC_ADC2_CLK_ENABLE

__HAL_RCC_ADC2_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_ADC3_CLK_ENABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_ADC3_CLK_DISABLE

__HAL_RCC_TIM9_CLK_ENABLE

__HAL_RCC_TIM10_CLK_ENABLE

__HAL_RCC_TIM11_CLK_ENABLE

__HAL_RCC_TIM9_CLK_DISABLE

__HAL_RCC_TIM10_CLK_DISABLE

__HAL_RCC_TIM11_CLK_DISABLE

APB2 Force Release Reset

__HAL_RCC_ADC2_FORCE_RESET

__HAL_RCC_ADC2_RELEASE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_ADC3_FORCE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_ADC3_RELEASE_RESET

__HAL_RCC_TIM9_FORCE_RESET

__HAL_RCC_TIM10_FORCE_RESET

__HAL_RCC_TIM11_FORCE_RESET

__HAL_RCC_TIM9_RELEASE_RESET

__HAL_RCC_TIM10_RELEASE_RESET

__HAL_RCC_TIM11_RELEASE_RESET

APB2 Peripheral Clock Enable Disable Status

__HAL_RCC_ADC2_IS_CLK_ENABLED

__HAL_RCC_ADC2_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_ADC3_IS_CLK_ENABLED

__HAL_RCC_ADC3_IS_CLK_DISABLED

__HAL_RCC_TIM9_IS_CLK_ENABLED

__HAL_RCC_TIM9_IS_CLK_DISABLED

__HAL_RCC_TIM10_IS_CLK_ENABLED

__HAL_RCC_TIM10_IS_CLK_DISABLED

__HAL_RCC_TIM11_IS_CLK_ENABLED

__HAL_RCC_TIM11_IS_CLK_DISABLED

HSE Configuration

__HAL_RCC_HSE_PREDIV_CONFIG **Description:**

- Macro to configure the External High Speed oscillator (HSE) Predivision factor for PLL.

Parameters:

- __HSE_PREDIV_VALUE__: specifies the division value applied to HSE. This parameter must be a number between RCC_HSE_PREDIV_DIV1 and RCC_HSE_PREDIV_DIV2.

Notes:

- Predivision factor can not be changed if PLL is used as system clock In this case, you have to select another source of the system clock, disable the PLL and then change the HSE predivision factor.

__HAL_RCC_HSE_GET_PREDIV

I2S2 Clock Source

RCC_I2S2CLKSOURCE_SYSCLK

I2S3 Clock Source

RCC_I2S3CLKSOURCE_SYSCLK

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

RCC_MCO1SOURCE_SYSCLK

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLLCLK

RCC Extended MCOx Clock Config

__HAL_RCC_MCO1_CONFIG **Description:**

- Macro to configure the MCO clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK No clock selected as MCO clock
  - RCC_MCO1SOURCE_SYSCLK System clock (SYSCLK) selected as MCO clock
  - RCC_MCO1SOURCE_HSI HSI selected as MCO clock
  - RCC_MCO1SOURCE_HSE HSE selected as MCO clock
  - RCC_MCO1SOURCE_PLLCLK PLL clock divided by 2 selected as MCO clock
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 No division applied on MCO clock source

Peripheral Clock Enable Disable

__HAL_RCC_DMA2_CLK_ENABLE

__HAL_RCC_DMA2_CLK_DISABLE

__HAL_RCC_FSMC_CLK_ENABLE

__HAL_RCC_FSMC_CLK_DISABLE

__HAL_RCC_SDIO_CLK_ENABLE

__HAL_RCC_SDIO_CLK_DISABLE

Peripheral Configuration

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock.

Parameters:

- __USBCLKSOURCE__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBCLKSOURCE_PLL PLL clock divided by 1 selected as USB clock
  - RCC_USBCLKSOURCE_PLL_DIV1_5 PLL clock divided by 1.5 selected as USB clock

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock (USBCLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_USBCLKSOURCE_PLL PLL clock divided by 1 selected as USB clock
  - RCC_USBCLKSOURCE_PLL_DIV1_5 PLL clock divided by 1.5 selected as USB

clock

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADCx clock (x=1 to 3 depending on devices).

Parameters:

- __ADCCLKSOURCE__: specifies the ADC clock source. This parameter can be one of the following values:
  - RCC_ADCPCLK2_DIV2 PCLK2 clock divided by 2 selected as ADC clock
  - RCC_ADCPCLK2_DIV4 PCLK2 clock divided by 4 selected as ADC clock
  - RCC_ADCPCLK2_DIV6 PCLK2 clock divided by 6 selected as ADC clock
  - RCC_ADCPCLK2_DIV8 PCLK2 clock divided by 8 selected as ADC clock

__HAL_RCC_GET_ADC_SOURCE

Description:

- Macro to get the ADC clock (ADCxCLK, x=1 to 3 depending on devices).

Return value:

- The: clock source can be one of the following values:
  - RCC_ADCPCLK2_DIV2 PCLK2 clock divided by 2 selected as ADC clock
  - RCC_ADCPCLK2_DIV4 PCLK2 clock divided by 4 selected as ADC clock
  - RCC_ADCPCLK2_DIV6 PCLK2 clock divided by 6 selected as ADC clock
  - RCC_ADCPCLK2_DIV8 PCLK2 clock divided by 8 selected as ADC clock

Periph Clock Selection

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_I2S2

RCC_PERIPHCLK_I2S3

RCC_PERIPHCLK_USB

PLL Multiplication Factor

RCC_PLL_MUL2

RCC_PLL_MUL3

RCC_PLL_MUL4

RCC_PLL_MUL5

RCC_PLL_MUL6

RCC_PLL_MUL7

RCC_PLL_MUL8

RCC_PLL_MUL9

RCC_PLL_MUL10

RCC_PLL_MUL11

RCC_PLL_MUL12

RCC_PLL_MUL13

RCC_PLL_MUL14

RCC_PLL_MUL15

RCC_PLL_MUL16

HSE Prediv1 Factor

RCC_HSE_PREDIV_DIV1

RCC_HSE_PREDIV_DIV2

USB Prescaler

RCC_USBCLKSOURCE_PLL

RCC_USBCLKSOURCE_PLL_DIV1_5

35 HAL RTC Generic Driver

35.1 RTC Firmware driver registers structures

35.1.1 RTC_TimeTypeDef

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

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 = 23
- ***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

35.1.2 RTC_AlarmTypeDef

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm ID (only 1 alarm ID for STM32F1). This parameter can be a value of ***RTC_Alarms_Definitions***

35.1.3 RTC_InitTypeDef

Data Fields

- ***uint32_t AsynchPrediv***
- ***uint32_t OutPut***

Field Documentation

- ***uint32_t RTC_InitTypeDef::AsynchPrediv***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFF or RTC_AUTO_1_SECOND If RTC_AUTO_1_SECOND is selected, AsynchPrediv will be set automatically to get 1sec timebase
- ***uint32_t RTC_InitTypeDef::OutPut***
  Specifies which signal will be routed to the RTC Tamper pin. This parameter can be a value of ***RTC_output_source_to_output_on_the_Tamper_pin***

35.1.4 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 (not necessary for HAL_RTC_SetDate). 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

35.1.5 RTC_HandleTypeDef

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***RTC_DateTypeDef DateToUpdate***
- ***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
- ***RTC_DateTypeDef RTC_HandleTypeDef::DateToUpdate***
  Current date set by user and updated automatically
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State***
  Time communication state

35.2 RTC Firmware driver API description

35.2.1 How to use this driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous prescaler to generate RTC 1Hz time base) 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.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper Level using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The TAMPER1 alternate function can be mapped to PC13

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEx_BKUPRead() function.

35.2.2 WARNING: Drivers Restrictions

RTC version used on STM32F1 families is version V1. All the features supported by V2 (other families) will be not supported on F1.

As on V2, main RTC features are managed by HW. But on F1, date feature is completely managed by SW.

Then, there are some restrictions compared to other families:

- Only format 24 hours supported in HAL (format 12 hours not supported)
- Date is saved in SRAM. Then, when MCU is in STOP or STANDBY mode, date will be lost. User should implement a way to save date before entering in low power mode (an example is provided with firmware package based on backup registers)
- Date is automatically updated each time a HAL_RTC_GetTime or HAL_RTC_GetDate is called.
- Alarm detection is limited to 1 day. It will expire only 1 time (no alarm repetition, need to program a new alarm)

35.2.3 Backup Domain Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

- The RTC
- The LSE oscillator
- PC13 I/O

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins are available:

- PC13 can be used as a Tamper pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

- PC13 can be used as the Tamper pin

35.2.4 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

35.2.5 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:

- Call the function HAL_RCCEx_PeriphCLKConfig in using RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSE)
- Enable the BKP clock in using __HAL_RCC_BKP_CLK_ENABLE()

35.2.6 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 RTC tamper 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.

35.2.7 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Asynchronous), 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 should be programmed to generate the RTC 1Hz time base.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by setting the CNF bit in the RTC_CRL register.
3. To read the calendar after wakeup from low power modes (Standby or Stop) the software must first wait for the RSF bit (Register Synchronized Flag) in the RTC_CRL register to be set by hardware. 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()*

35.2.8 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()*

35.2.9 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_GetAlarm()*
- *HAL_RTC_DeactivateAlarm()*
- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

35.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

35.2.11 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()*

35.2.12 Detailed description of functions

HAL_RTC_Init

Function name	**HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)**
Function description	Initializes the RTC peripheral.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• **HAL:** status

HAL_RTC_DeInit

Function name **HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)**

Function description DeInitializes the RTC peripheral.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **HAL:** status

Notes
- This function does not reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name **void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)**

Function description Initializes the RTC MSP.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTC_MspDeInit

Function name **void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)**

Function description DeInitializes the RTC MSP.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTC_SetTime

Function name **HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**

Function description Sets RTC current time.

Parameters
- **hrtc:** 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:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

HAL_RTC_GetTime

Function name **HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**

Function description Gets RTC current time.

Parameters
- **hrtc:** 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:
  – RTC_FORMAT_BIN: Binary data format
  – RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

HAL_RTC_SetDate

Function name **HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**

Function description Sets RTC current date.

Parameters
- **hrtc:** 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:
  – RTC_FORMAT_BIN: Binary data format
  – RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

HAL_RTC_GetDate

Function name **HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**

Function description Gets RTC current date.

Parameters
- **hrtc:** 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:
  – RTC_FORMAT_BIN: Binary data format
  – RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

HAL_RTC_SetAlarm

Function name **HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**

Function description Sets the specified RTC Alarm.

Parameters
- **hrtc:** 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:
  – RTC_FORMAT_BIN: Binary data format

– RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

HAL_RTC_SetAlarm_IT

Function name **HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**

Function description Sets the specified RTC Alarm with Interrupt.

Parameters
- **hrtc:** 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:
  – RTC_FORMAT_BIN: Binary data format
  – RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

Notes
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

HAL_RTC_DeactivateAlarm

Function name **HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)**

Function description Deactive the specified RTC Alarm.

Parameters
- **hrtc:** 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

Return values
- **HAL:** status

HAL_RTC_GetAlarm

Function name **HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)**

Function description Gets the RTC Alarm value and masks.

Parameters
- **hrtc:** 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: Alarm
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  – RTC_FORMAT_BIN: Binary data format
  – RTC_FORMAT_BCD: BCD data format

Return values
- **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name **void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)**

Function description This function handles Alarm interrupt request.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTC_PollForAlarmAEvent

Function name **HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**

Function description This function handles AlarmA Polling request.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_RTC_AlarmAEventCallback

Function name **void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)**

Function description Alarm A callback.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTC_GetState

Function name **HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)**

Function description Returns the RTC state.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **HAL:** state

HAL_RTC_WaitForSynchro

Function name **HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)**

Function description Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **HAL:** status

Notes
- This function must be called before any read operation after an APB reset or an APB clock stop.

35.3 RTC Firmware driver defines

35.3.1 RTC

Alarms Definitions

RTC_ALARM_A Specify alarm ID (mainly for legacy purposes)

Automatic calculation of prediv for 1sec timebase

RTC_AUTO_1_SECOND

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:
- Reset RTC handle state.

Parameters:
- __HANDLE__: RTC handle.

Return value:
- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:
- Disable the write protection for RTC registers.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:
- Enable the write protection for RTC registers.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_ALARM_ENABLE_IT

Description:
- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  – RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  – RTC_IT_ALRA: Alarm A interrupt

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 be checked This parameter can be:
  – RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_ALRAF

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA: Alarm A interrupt

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

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on ALARM Exti Line 17.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_IT

Description:

- Disable interrupt on ALARM Exti Line 17.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

Description:

- Enable event on ALARM Exti Line 17.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Description:

- Disable event on ALARM Exti Line 17.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE

Description:

- ALARM EXTI line configuration: set falling edge trigger.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the ALARM Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE

Description:

- ALARM EXTI line configuration: set rising edge trigger.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the ALARM Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- ALARM EXTI line configuration: set rising & falling edge trigger.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the ALARM Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_GET_FLAG

Description:

- Check whether the specified ALARM EXTI interrupt flag is set or not.

Return value:

- EXTI: ALARM Line Status.

__HAL_RTC_ALARM_EXTI_CLEAR_FLAG

Description:

- Clear the ALARM EXTI flag.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

RTC EXTI Line event

RTC_EXTI_LINE_ALARM_EVENT	External interrupt line 17 Connected to the RTC Alarm event

Flags Definitions

RTC_FLAG_RTOFF	RTC Operation OFF flag
RTC_FLAG_RSF	Registers Synchronized flag
RTC_FLAG_OW	Overflow flag
RTC_FLAG_ALRAF	Alarm flag
RTC_FLAG_SEC	Second flag
RTC_FLAG_TAMP1F	Tamper Interrupt Flag

Input Parameter Format

RTC_FORMAT_BIN

RTC_FORMAT_BCD

Interrupts Definitions

RTC_IT_OW	Overflow interrupt
RTC_IT_ALRA	Alarm interrupt
RTC_IT_SEC	Second interrupt
RTC_IT_TAMP1	TAMPER Pin interrupt enable

Month Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

Output source to output on the Tamper pin

RTC_OUTPUTSOURCE_NONE	No output on the TAMPER pin
RTC_OUTPUTSOURCE_CALIBCLOCK	RTC clock with a frequency divided by 64 on the TAMPER pin
RTC_OUTPUTSOURCE_ALARM	Alarm pulse signal on the TAMPER pin
RTC_OUTPUTSOURCE_SECOND	Second pulse signal on the TAMPER pin

Default Timeout Value

RTC_TIMEOUT_VALUE

WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

36 HAL RTC Extension Driver

36.1 RTCEx Firmware driver registers structures

36.1.1 RTC_TamperTypeDef

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins_Definitions***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger_Definitions***

36.2 RTCEx Firmware driver API description

36.2.1 RTC Tamper functions

This section provides functions allowing to configure Tamper feature

This section contains the following APIs:

- ***HAL_RTCEx_SetTamper()***
- ***HAL_RTCEx_SetTamper_IT()***
- ***HAL_RTCEx_DeactivateTamper()***
- ***HAL_RTCEx_TamperIRQHandler()***
- ***HAL_RTCEx_Tamper1EventCallback()***
- ***HAL_RTCEx_PollForTamper1Event()***

36.2.2 RTC Second functions

This section provides functions implementing second interupt handlers

This section contains the following APIs:

- ***HAL_RTCEx_SetSecond_IT()***
- ***HAL_RTCEx_DeactivateSecond()***
- ***HAL_RTCEx_RTCIRQHandler()***
- ***HAL_RTCEx_RTCEventCallback()***
- ***HAL_RTCEx_RTCEventErrorCallback()***

36.2.3 Extension Peripheral Control functions

This subsection provides functions allowing to

- Writes a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Sets the Smooth calibration parameters.

This section contains the following APIs:

- *HAL_RTCEx_BKUPWrite()*
- *HAL_RTCEx_BKUPRead()*
- *HAL_RTCEx_SetSmoothCalib()*

36.2.4 Detailed description of functions

HAL_RTCEx_SetTamper

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function description	Sets Tamper.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **sTamper:** Pointer to Tamper Structure.
Return values	• **HAL:** status
Notes	• By calling this API we disable the tamper interrupt for all tampers. • Tamper can be enabled only if ASOE and CCO bit are reset

HAL_RTCEx_SetTamper_IT

Function name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function description	Sets Tamper with interrupt.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **sTamper:** Pointer to RTC Tamper.
Return values	• **HAL:** status
Notes	• By calling this API we force the tamper interrupt for all tampers. • Tamper can be enabled only if ASOE and CCO bit are reset

HAL_RTCEx_DeactivateTamper

Function name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)**
Function description	Deactivates Tamper.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **Tamper:** Selected tamper pin. This parameter can be a value of Tamper Pins Definitions
Return values	• **HAL:** status

HAL_RTCEx_TamperIRQHandler

Function name	**void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)**

Function description This function handles Tamper interrupt request.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTCEx_Tamper1EventCallback

Function name **void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)**

Function description Tamper 1 callback.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTCEx_PollForTamper1Event

Function name **HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**

Function description This function handles Tamper1 Polling.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_RTCEx_SetSecond_IT

Function name **HAL_StatusTypeDef HAL_RTCEx_SetSecond_IT (RTC_HandleTypeDef * hrtc)**

Function description Sets Interrupt for second.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **HAL:** status

HAL_RTCEx_DeactivateSecond

Function name **HAL_StatusTypeDef HAL_RTCEx_DeactivateSecond (RTC_HandleTypeDef * hrtc)**

Function description Deactivates Second.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **HAL:** status

HAL_RTCEx_RTCIRQHandler

Function name **void HAL_RTCEx_RTCIRQHandler (RTC_HandleTypeDef * hrtc)**

Function description This function handles second interrupt request.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTCEx_RTCEventCallback

Function name **void HAL_RTCEx_RTCEventCallback (RTC_HandleTypeDef * hrtc)**

Function description Second event callback.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTCEx_RTCEventErrorCallback

Function name **void HAL_RTCEx_RTCEventErrorCallback (RTC_HandleTypeDef * hrtc)**

Function description Second event error callback.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values
- **None:**

HAL_RTCEx_BKUPWrite

Function name **void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)**

Function description Writes a data in a specified RTC Backup data register.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 1 to 10 (or 42) to specify the register (depending devices).
- **Data:** Data to be written in the specified RTC Backup data register.

Return values
- **None:**

HAL_RTCEx_BKUPRead

Function name **uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)**

Function description Reads data from the specified RTC Backup data Register.

Parameters
- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 1 to 10 (or 42) to specify the register (depending devices).

Return values
- **Read:** value

HAL_RTCEx_SetSmoothCalib

Function name **HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmouthCalibMinusPulsesValue)**

Function description Sets the Smooth calibration parameters.

Parameters
- **hrtc:** RTC handle
- **SmoothCalibPeriod:** Not used (only present for compatibility with another families)
- **SmoothCalibPlusPulses:** Not used (only present for compatibility with another families)
- **SmouthCalibMinusPulsesValue:** specifies the RTC Clock Calibration value. This parameter must be a number between 0 and 0x7F.

Return values
- **HAL:** status

36.3 RTCEx Firmware driver defines

36.3.1 RTCEx

Alias define maintained for legacy

HAL_RTCEx_TamperTimeStampIRQHandler

Backup Registers Definitions

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
RTC_BKP_DR32
RTC_BKP_DR33
RTC_BKP_DR34
RTC_BKP_DR35
RTC_BKP_DR36
RTC_BKP_DR37
RTC_BKP_DR38
RTC_BKP_DR39
RTC_BKP_DR40
RTC_BKP_DR41
RTC_BKP_DR42

RTCEx Exported Macros

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled This parameter can be any combination of the following values:
  – RTC_IT_TAMP1: Tamper A interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- Disable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  – RTC_IT_TAMP1: Tamper A 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 sources to be checked. This parameter can be:
  – RTC_IT_TAMP1

Return value:

- None

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_TAMP1F

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be checked. This parameter can be:
  – RTC_IT_TAMP1

Return value:

- None

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_TAMP1F

Return value:

- None

__HAL_RTC_SECOND_ENABLE_IT

Description:

- Enable the RTC Second interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Second interrupt sources to be enabled This parameter can be any combination of the following values:
  – RTC_IT_SEC: Second A interrupt

Return value:

- None

__HAL_RTC_SECOND_DISABLE_IT

Description:

- Disable the RTC Second interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Second interrupt sources to be disabled. This parameter can be any combination of the following values:
  – RTC_IT_SEC: Second A interrupt

Return value:

- None

__HAL_RTC_SECOND_GET_IT_SOURCE

Description:

- Check whether the specified RTC Second interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Second interrupt sources to be enabled or disabled. This parameter can be:
  – RTC_IT_SEC: Second A interrupt

Return value:

- None

__HAL_RTC_SECOND_GET_FLAG

Description:

- Get the selected RTC Second's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Second Flag sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_SEC

Return value:

- None

__HAL_RTC_SECOND_CLEAR_FLAG

Description:

- Clear the RTC Second's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Second Flag sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_SEC

Return value:

- None

__HAL_RTC_OVERFLOW_ENABLE_IT

Description:

- Enable the RTC Overflow interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Overflow interrupt sources to be enabled This parameter can be any combination of the following values:
  – RTC_IT_OW: Overflow A interrupt

Return value:

- None

__HAL_RTC_OVERFLOW_DISABLE_IT

Description:

- Disable the RTC Overflow interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Overflow interrupt sources to be disabled. This parameter can be any combination of the following values:
  – RTC_IT_OW: Overflow A interrupt

Return value:

- None

__HAL_RTC_OVERFLOW_GET_IT_SOURCE

Description:

- Check whether the specified RTC Overflow interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Overflow interrupt sources to be enabled or disabled. This parameter can be:
  – RTC_IT_OW: Overflow A interrupt

Return value:

- None

__HAL_RTC_OVERFLOW_GET_FLAG

Description:

- Get the selected RTC Overflow's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Overflow Flag sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_OW

Return value:

- None

__HAL_RTC_OVERFLOW_CLEAR_FLAG

Description:

- Clear the RTC Overflow's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Overflow Flag sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_OW

Return value:

- None

Private macros to check input parameters

IS_RTC_TAMPER

IS_RTC_TAMPER_TRIGGER

IS_RTC_BKP

IS_RTC_SMOOTH_CALIB_MINUS

Tamper Pins Definitions

RTC_TAMPER_1	Select tamper to be enabled (mainly for legacy purposes)

Tamper Trigger Definitions

RTC_TAMPERTRIGGER_LOWLEVEL	A high level on the TAMPER pin resets all data backup registers (if TPE bit is set)
RTC_TAMPERTRIGGER_HIGHLEVEL	A low level on the TAMPER pin resets all data backup registers (if TPE bit is set)

37 HAL SD Generic Driver

37.1 SD Firmware driver registers structures

37.1.1 HAL_SD_CardInfoTypeDef

Data Fields

- ***uint32_t CardType***
- ***uint32_t CardVersion***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***

Field Documentation

- ***uint32_t HAL_SD_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_SD_CardInfoTypeDef::CardVersion***
  Specifies the card version
- ***uint32_t HAL_SD_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_SD_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes

37.1.2 SD_HandleTypeDef

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***uint32_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint32_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_SD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***DMA_HandleTypeDef * hdmarx***
- ***DMA_HandleTypeDef * hdmatx***
- ***HAL_SD_CardInfoTypeDef SdCard***

- ***uint32_t CSD***
- ***uint32_t CID***

Field Documentation

- ***SD_TypeDef* SD_HandleTypeDef::Instance***
  SD registers base address
- ***SD_InitTypeDef SD_HandleTypeDef::Init***
  SD required parameters
- ***HAL_LockTypeDef SD_HandleTypeDef::Lock***
  SD locking object
- ***uint32_t* SD_HandleTypeDef::pTxBuffPtr***
  Pointer to SD Tx transfer Buffer
- ***uint32_t SD_HandleTypeDef::TxXferSize***
  SD Tx Transfer size
- ***uint32_t* SD_HandleTypeDef::pRxBuffPtr***
  Pointer to SD Rx transfer Buffer
- ***uint32_t SD_HandleTypeDef::RxXferSize***
  SD Rx Transfer size
- ***__IO uint32_t SD_HandleTypeDef::Context***
  SD transfer context
- ***__IO HAL_SD_StateTypeDef SD_HandleTypeDef::State***
  SD card State
- ***__IO uint32_t SD_HandleTypeDef::ErrorCode***
  SD Card Error codes
- ***DMA_HandleTypeDef* SD_HandleTypeDef::hdmarx***
  SD Rx DMA handle parameters
- ***DMA_HandleTypeDef* SD_HandleTypeDef::hdmatx***
  SD Tx DMA handle parameters
- ***HAL_SD_CardInfoTypeDef SD_HandleTypeDef::SdCard***
  SD Card information
- ***uint32_t SD_HandleTypeDef::CSD[4]***
  SD card specific data table
- ***uint32_t SD_HandleTypeDef::CID[4]***
  SD card identification number table

37.1.3 HAL_SD_CardCSDTypeDef

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***
- ***__IO uint8_t TAAC***
- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***

- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGrouop***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_SD_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_SD_CardCSDTypeDef::DeviceSize***
  Device Size

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrSpeedFact***
  Write speed factor
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormatGrouop***
  File format group
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved4***
  Always 1

37.1.4 HAL_SD_CardCIDTypeDef

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***

- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2***
  Always 1

37.1.5 HAL_SD_CardStatusTypeDef

Data Fields

- ***__IO uint8_t DataBusWidth***
- ***__IO uint8_t SecuredMode***
- ***__IO uint16_t CardType***
- ***__IO uint32_t ProtectedAreaSize***
- ***__IO uint8_t SpeedClass***
- ***__IO uint8_t PerformanceMove***
- ***__IO uint8_t AllocationUnitSize***
- ***__IO uint16_t EraseSize***
- ***__IO uint8_t EraseTimeout***
- ***__IO uint8_t EraseOffset***

Field Documentation

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::DataBusWidth***
  Shows the currently defined data bus width
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SecuredMode***
  Card is in secured mode of operation
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::CardType***
  Carries information about card type
- ***__IO uint32_t HAL_SD_CardStatusTypeDef::ProtectedAreaSize***
  Carries information about the capacity of protected area

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SpeedClass***
  Carries information about the speed class of the card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::PerformanceMove***
  Carries information about the card's performance move
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::AllocationUnitSize***
  Carries information about the card's allocation unit size
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::EraseSize***
  Determines the number of AUs to be erased in one operation
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseTimeout***
  Determines the timeout for any number of AU erase
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseOffset***
  Carries information about the erase offset

37.2 SD Firmware driver API description

37.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDIO and GPIO) are performed by the user in HAL_SD_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDIO memories which uses the HAL SDIO driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDIO low level resources by implement the HAL_SD_MspInit() API:
   a. Enable the SDIO interface clock using __HAL_RCC_SDIO_CLK_ENABLE();
   b. SDIO pins configuration for SD card
      – Enable the clock for the SDIO GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      – Configure these SDIO pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. DMA Configuration if you need to use DMA process (HAL_SD_ReadBlocks_DMA() and HAL_SD_WriteBlocks_DMA() APIs).
      – Enable the DMAx interface clock using __HAL_RCC_DMAx_CLK_ENABLE();
      – Configure the DMA using the function HAL_DMA_Init() with predeclared and filled.
   d. NVIC configuration if you need to use interrupt process when using DMA transfer.
      – Configure the SDIO and DMA interrupt priorities using functions HAL_NVIC_SetPriority(); DMA priority is superior to SDIO's priority
      – Enable the NVIC DMA and SDIO IRQs using function HAL_NVIC_EnableIRQ()
      – SDIO interrupts are managed using the macros __HAL_SD_ENABLE_IT() and __HAL_SD_DISABLE_IT() inside the communication process.
      – SDIO interrupts pending bits are managed using the macros __HAL_SD_GET_IT() and __HAL_SD_CLEAR_IT()
   e. NVIC configuration if you need to use interrupt process (HAL_SD_ReadBlocks_IT() and HAL_SD_WriteBlocks_IT() APIs).
      – Configure the SDIO interrupt priorities using function HAL_NVIC_SetPriority();
      – Enable the NVIC SDIO IRQs using function HAL_NVIC_EnableIRQ()
      – SDIO interrupts are managed using the macros __HAL_SD_ENABLE_IT() and __HAL_SD_DISABLE_IT() inside the communication process.

  - SDIO interrupts pending bits are managed using the macros __HAL_SD_GET_IT() and __HAL_SD_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 SDIO IP(STM32 side) and the SD Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDIO peripheral interface with defaullt configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO_CK) is computed as follows: SDIO_CK = SDIOCLK / (ClockDiv + 2) In initialization mode and according to the SD Card standard, make sure that the SDIO_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDIO_Init() and SDIO_PowerState_ON() SDIO low level APIs.
2. Initialize the SD card. The API used is HAL_SD_InitCard(). This phase allows the card initialization and identification and check the SD Card type (Standard Capacity or High Capacity) The initialization flow is compatible with SD standard. This API (HAL_SD_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the SD Card Data transfer frequency. By Default, the card transfer frequency is set to 24MHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDIO_CK frequency doesn't exceed 25MHz and 50MHz in High-speed mode switch. To be able to use a frequency higher than 24MHz, you should use the SDIO peripheral in bypass mode. Refer to the corresponding reference manual for more details.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function HAL_SD_ReadBlocks(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can read from SD card in DMA mode by using function HAL_SD_ReadBlocks_DMA(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Rx interrupt event.
- You can read from SD card in Interrupt mode by using function HAL_SD_ReadBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Rx interrupt event.

SD Card Write operation

- You can write to SD card in polling mode by using function HAL_SD_WriteBlocks(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can write to SD card in DMA mode by using function HAL_SD_WriteBlocks_DMA(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Tx interrupt event.
- You can write to SD card in Interrupt mode by using function HAL_SD_WriteBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Tx interrupt event.

SD card status

- The SD Status contains status bits that are related to the SD Memory Card proprietary features. To get SD card status use the HAL_SD_GetCardStatus().

SD card information

- To get SD card information, you can use the function HAL_SD_GetCardInfo(). It returns useful information about the SD card such as block size, card type, block number ...

SD card CSD register

- The HAL_SD_GetCardCSD() API allows to get the parameters of the CSD register. Some of the CSD parameters are useful for card initialization and identification.

SD card CID register

- The HAL_SD_GetCardCID() API allows to get the parameters of the CID register. Some of the CSD parameters are useful for card initialization and identification.

SD HAL driver macros list

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

- __HAL_SD_ENABLE : Enable the SD device
- __HAL_SD_DISABLE : Disable the SD device
- __HAL_SD_DMA_ENABLE: Enable the SDIO DMA transfer
- __HAL_SD_DMA_DISABLE: Disable the SDIO DMA transfer
- __HAL_SD_ENABLE_IT: Enable the SD device interrupt
- __HAL_SD_DISABLE_IT: Disable the SD device interrupt
- __HAL_SD_GET_FLAG:Check whether the specified SD flag is set or not
- __HAL_SD_CLEAR_FLAG: Clear the SD's pending flags



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

37.2.2 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_InitCard()***
- ***HAL_SD_DeInit()***
- ***HAL_SD_MspInit()***
- ***HAL_SD_MspDeInit()***

37.2.3 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_IT()***
- ***HAL_SD_WriteBlocks_IT()***
- ***HAL_SD_ReadBlocks_DMA()***
- ***HAL_SD_WriteBlocks_DMA()***
- ***HAL_SD_Erase()***
- ***HAL_SD_IRQHandler()***
- ***HAL_SD_GetState()***
- ***HAL_SD_GetError()***
- ***HAL_SD_TxCpltCallback()***
- ***HAL_SD_RxCpltCallback()***
- ***HAL_SD_ErrorCallback()***
- ***HAL_SD_AbortCallback()***

37.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations and get the related information

This section contains the following APIs:

- ***HAL_SD_GetCardCID()***
- ***HAL_SD_GetCardCSD()***
- ***HAL_SD_GetCardStatus()***
- ***HAL_SD_GetCardInfo()***
- ***HAL_SD_ConfigWideBusOperation()***
- ***HAL_SD_GetCardState()***
- ***HAL_SD_Abort()***
- ***HAL_SD_Abort_IT()***

37.2.5 Detailed description of functions

HAL_SD_Init

Function name **HAL_StatusTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd)**

Function description Initializes the SD according to the specified parameters in the SD_HandleTypeDef and create the associated handle.

Parameters
- **hsd:** Pointer to the SD handle

Return values
- **HAL:** status

HAL_SD_InitCard

Function name **HAL_StatusTypeDef HAL_SD_InitCard (SD_HandleTypeDef * hsd)**

Function description Initializes the SD Card.

Parameters
- **hsd:** Pointer to SD handle

Return values
- **HAL:** status

Notes
- This function initializes the SD card. It could be used when a card re-initialization is needed.

HAL_SD_DeInit

Function name **HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)**

Function description De-Initializes the SD card.

Parameters
- **hsd:** Pointer to SD handle

Return values
- **HAL:** status

HAL_SD_MspInit

Function name **void HAL_SD_MspInit (SD_HandleTypeDef * hsd)**

Function description Initializes the SD MSP.

Parameters
- **hsd:** Pointer to SD handle

Return values
- **None:**

HAL_SD_MspDeInit

Function name **void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)**

Function description De-Initialize SD MSP.

Parameters
- **hsd:** Pointer to SD handle

Return values
- **None:**

HAL_SD_ReadBlocks

Function name **HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd,**

uint32_t NumberOfBlocks, uint32_t Timeout)

Function description: Reads block(s) from a specified address in a card.

Parameters:
- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of SD blocks to read
- **Timeout:** Specify timeout value

Return values:
- **HAL:** status

Notes:
- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_WriteBlocks

Function name: **HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)**

Function description: Allows to write block(s) to a specified address in a card.

Parameters:
- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of SD blocks to write
- **Timeout:** Specify timeout value

Return values:
- **HAL:** status

Notes:
- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_Erase

Function name: **HAL_StatusTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint32_t BlockStartAdd, uint32_t BlockEndAdd)**

Function description: Erases the specified memory area of the given SD card.

Parameters:
- **hsd:** Pointer to SD handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values:
- **HAL:** status

Notes:
- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_ReadBlocks_IT

Function name: **HAL_StatusTypeDef HAL_SD_ReadBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)**

Function description: Reads block(s) from a specified address in a card.

Parameters:
- **hsd:** Pointer to SD handle

- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values
- **HAL:** status

Notes
- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_IT

Function name **HAL_StatusTypeDef HAL_SD_WriteBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)**

Function description Writes block(s) to a specified address in a card.

Parameters
- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values
- **HAL:** status

Notes
- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Tx interrupt event.

HAL_SD_ReadBlocks_DMA

Function name **HAL_StatusTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)**

Function description Reads block(s) from a specified address in a card.

Parameters
- **hsd:** Pointer SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values
- **HAL:** status

Notes
- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_DMA

Function name **HAL_StatusTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd,**

uint32_t NumberOfBlocks)

Function description	Writes block(s) to a specified address in a card.
Parameters	• **hsd:** Pointer to SD handle • **pData:** Pointer to the buffer that will contain the data to transmit • **BlockAdd:** Block Address where data will be written • **NumberOfBlocks:** Number of blocks to write
Return values	• **HAL:** status
Notes	• This API should be followed by a check on the card state through HAL_SD_GetCardState(). • You could also check the DMA transfer process through the SD Tx interrupt event.

HAL_SD_IRQHandler

Function name	**void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)**
Function description	This function handles SD card interrupt request.
Parameters	• **hsd:** Pointer to SD handle
Return values	• **None:**

HAL_SD_TxCpltCallback

Function name	**void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)**
Function description	Tx Transfer completed callbacks.
Parameters	• **hsd:** Pointer to SD handle
Return values	• **None:**

HAL_SD_RxCpltCallback

Function name	**void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)**
Function description	Rx Transfer completed callbacks.
Parameters	• **hsd:** Pointer SD handle
Return values	• **None:**

HAL_SD_ErrorCallback

Function name	**void HAL_SD_ErrorCallback (SD_HandleTypeDef * hsd)**
Function description	SD error callbacks.
Parameters	• **hsd:** Pointer SD handle
Return values	• **None:**

HAL_SD_AbortCallback

Function name	**void HAL_SD_AbortCallback (SD_HandleTypeDef * hsd)**

Function description SD Abort callbacks.

Parameters
- **hsd:** Pointer SD handle

Return values
- **None:**

HAL_SD_ConfigWideBusOperation

Function name **HAL_StatusTypeDef HAL_SD_ConfigWideBusOperation (SD_HandleTypeDef * hsd, uint32_t WideMode)**

Function description Enables wide bus operation for the requested card if supported by card.

Parameters
- **hsd:** Pointer to SD handle
- **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values:
  - SDIO_BUS_WIDE_8B: 8-bit data transfer
  - SDIO_BUS_WIDE_4B: 4-bit data transfer
  - SDIO_BUS_WIDE_1B: 1-bit data transfer

Return values
- **HAL:** status

HAL_SD_SendSDStatus

Function name **HAL_StatusTypeDef HAL_SD_SendSDStatus (SD_HandleTypeDef * hsd, uint32_t * pSDstatus)**

Function description

HAL_SD_GetCardState

Function name **HAL_SD_CardStateTypeDef HAL_SD_GetCardState (SD_HandleTypeDef * hsd)**

Function description Gets the current sd card data state.

Parameters
- **hsd:** pointer to SD handle

Return values
- **Card:** state

HAL_SD_GetCardCID

Function name **HAL_StatusTypeDef HAL_SD_GetCardCID (SD_HandleTypeDef * hsd, HAL_SD_CardCIDTypeDef * pCID)**

Function description Returns information the information of the card which are stored on the CID register.

Parameters
- **hsd:** Pointer to SD handle
- **pCID:** Pointer to a HAL_SD_CIDTypeDef structure that contains all CID register parameters

Return values
- **HAL:** status

HAL_SD_GetCardCSD

Function name **HAL_StatusTypeDef HAL_SD_GetCardCSD (SD_HandleTypeDef * hsd, HAL_SD_CardCSDTypeDef ***

pCSD)

Function description: Returns information the information of the card which are stored on the CSD register.

Parameters:
- **hsd:** Pointer to SD handle
- **pCSD:** Pointer to a HAL_SD_CardCSDTypeDef structure that contains all CSD register parameters

Return values:
- **HAL:** status

HAL_SD_GetCardStatus

Function name: **HAL_StatusTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pStatus)**

Function description: Gets the SD status info.

Parameters:
- **hsd:** Pointer to SD handle
- **pStatus:** Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information

Return values:
- **HAL:** status

HAL_SD_GetCardInfo

Function name: **HAL_StatusTypeDef HAL_SD_GetCardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)**

Function description: Gets the SD card info.

Parameters:
- **hsd:** Pointer to SD handle
- **pCardInfo:** Pointer to the HAL_SD_CardInfoTypeDef structure that will contain the SD card status information

Return values:
- **HAL:** status

HAL_SD_GetState

Function name: **HAL_SD_StateTypeDef HAL_SD_GetState (SD_HandleTypeDef * hsd)**

Function description: return the SD state

Parameters:
- **hsd:** Pointer to sd handle

Return values:
- **HAL:** state

HAL_SD_GetError

Function name: **uint32_t HAL_SD_GetError (SD_HandleTypeDef * hsd)**

Function description: Return the SD error code.

Parameters:
- **hsd:** : Pointer to a SD_HandleTypeDef structure that contains the configuration information.

Return values:
- **SD:** Error Code

HAL_SD_Abort

Function name	**HAL_StatusTypeDef HAL_SD_Abort (SD_HandleTypeDef * hsd)**
Function description	Abort the current transfer and disable the SD.
Parameters	• **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.
Return values	• **HAL:** status

HAL_SD_Abort_IT

Function name	**HAL_StatusTypeDef HAL_SD_Abort_IT (SD_HandleTypeDef * hsd)**
Function description	Abort the current transfer and disable the SD (IT mode).
Parameters	• **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.
Return values	• **HAL:** status

37.3 SD Firmware driver defines

37.3.1 SD

SD Error status enumeration Structure definition

HAL_SD_ERROR_NONE	No error
HAL_SD_ERROR_CMD_CRC_FAIL	Command response received (but CRC check failed)
HAL_SD_ERROR_DATA_CRC_FAIL	Data block sent/received (CRC check failed)
HAL_SD_ERROR_CMD_RSP_TIMEOUT	Command response timeout
HAL_SD_ERROR_DATA_TIMEOUT	Data timeout
HAL_SD_ERROR_TX_UNDERRUN	Transmit FIFO underrun
HAL_SD_ERROR_RX_OVERRUN	Receive FIFO overrun
HAL_SD_ERROR_ADDR_MISALIGNED	Misaligned address
HAL_SD_ERROR_BLOCK_LEN_ERR	Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length
HAL_SD_ERROR_ERASE_SEQ_ERR	An error in the sequence of erase command occurs
HAL_SD_ERROR_BAD_ERASE_PARAM	An invalid selection for erase groups
HAL_SD_ERROR_WRITE_PROT_VIOLATION	Attempt to program a write protect block
HAL_SD_ERROR_LOCK_UNLOCK_FAILED	Sequence or password error has been detected in unlock command or if there was an attempt to access a

	locked card
HAL_SD_ERROR_COM_CRC_FAILED	CRC check of the previous command failed
HAL_SD_ERROR_ILLEGAL_CMD	Command is not legal for the card state
HAL_SD_ERROR_CARD_ECC_FAILED	Card internal ECC was applied but failed to correct the data
HAL_SD_ERROR_CC_ERR	Internal card controller error
HAL_SD_ERROR_GENERAL_UNKNOWN_ERR	General or unknown error
HAL_SD_ERROR_STREAM_READ_UNDERRUN	The card could not sustain data reading in stream rmode
HAL_SD_ERROR_STREAM_WRITE_OVERRUN	The card could not sustain data programming in stream mode
HAL_SD_ERROR_CID_CSD_OVERWRITE	CID/CSD overwrite error
HAL_SD_ERROR_WP_ERASE_SKIP	Only partial address space was erased
HAL_SD_ERROR_CARD_ECC_DISABLED	Command has been executed without using internal ECC
HAL_SD_ERROR_ERASE_RESET	Erase sequence was cleared before executing because an out of erase sequence command was received
HAL_SD_ERROR_AKE_SEQ_ERR	Error in sequence of authentication
HAL_SD_ERROR_INVALID_VOLTRANGE	Error in case of invalid voltage range
HAL_SD_ERROR_ADDR_OUT_OF_RANGE	Error when addressed block is out of range
HAL_SD_ERROR_REQUEST_NOT_APPLICABLE	Error when command request is not applicable
HAL_SD_ERROR_PARAM	the used parameter is not valid
HAL_SD_ERROR_UNSUPPORTED_FEATURE	Error when feature is not insupported
HAL_SD_ERROR_BUSY	Error when transfer process is busy
HAL_SD_ERROR_DMA	Error while DMA transfer
HAL_SD_ERROR_TIMEOUT	Timeout error

SD context enumeration

SD_CONTEXT_NONE	None
SD_CONTEXT_READ_SINGLE_BLOCK	Read single block operation
SD_CONTEXT_READ_MULTIPLE_BLOCK	Read multiple blocks operation
SD_CONTEXT_WRITE_SINGLE_BLOCK	Write single block operation
SD_CONTEXT_WRITE_MULTIPLE_BLOCK	Write multiple blocks operation
SD_CONTEXT_IT	Process in Interrupt mode
SD_CONTEXT_DMA	Process in DMA mode

SD Supported Memory Cards

CARD_SDSC

CARD_SDHC_SDXC

CARD_SECURED

SD Supported Version

CARD_V1_X

CARD_V2_X

Exported Constants

BLOCKSIZE Block size is 512 bytes

SD Exported Macros

__HAL_SD_ENABLE **Description:**

- Enable the SD device.

Return value:

- None

__HAL_SD_DISABLE **Description:**

- Disable the SD device.

Return value:

- None

__HAL_SD_DMA_ENABLE **Description:**

- Enable the SDMMC DMA transfer.

Return value:

- None

__HAL_SD_DMA_DISABLE **Description:**

- Disable the SDMMC DMA transfer.

Return value:

- None

__HAL_SD_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:
  - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO_IT_CTIMEOUT: Command response timeout interrupt

- SDIO_IT_DTIMEOUT: Data timeout interrupt
- SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
- SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDIO_IT_CMDSENT: Command sent (no response required) interrupt
- SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO_IT_CMDACT: Command transfer in progress interrupt
- SDIO_IT_TXACT: Data transmit in progress interrupt
- SDIO_IT_RXACT: Data receive in progress interrupt
- SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO_IT_TXFIFOF: Transmit FIFO full interrupt
- SDIO_IT_RXFIFOF: Receive FIFO full interrupt
- SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
- SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
- SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
- SDIO_IT_SDIOIT: SD I/O interrupt received interrupt

Return value:

- None

__HAL_SD_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:
  - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO_IT_CTIMEOUT: Command response timeout interrupt
  - SDIO_IT_DTIMEOUT: Data timeout interrupt
  - SDIO_IT_TXUNDERR: Transmit FIFO underrun

error interrupt
  - SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDIO_IT_CMDSENT: Command sent (no response required) interrupt
  - SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
  - SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDIO_IT_CMDACT: Command transfer in progress interrupt
  - SDIO_IT_TXACT: Data transmit in progress interrupt
  - SDIO_IT_RXACT: Data receive in progress interrupt
  - SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDIO_IT_TXFIFOF: Transmit FIFO full interrupt
  - SDIO_IT_RXFIFOF: Receive FIFO full interrupt
  - SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
  - SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
  - SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
  - SDIO_IT_SDIOIT: SD I/O interrupt received interrupt

Return value:

- None

__HAL_SD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: SD Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDIO_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDIO_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDIO_FLAG_CTIMEOUT: Command response timeout
  - SDIO_FLAG_DTIMEOUT: Data timeout
  - SDIO_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDIO_FLAG_RXOVERR: Received FIFO overrun error

- SDIO_FLAG_CMDREND: Command response received (CRC check passed)
- SDIO_FLAG_CMDSENT: Command sent (no response required)
- SDIO_FLAG_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
- SDIO_FLAG_DBCKEND: Data block sent/received (CRC check passed)
- SDIO_FLAG_CMDACT: Command transfer in progress
- SDIO_FLAG_TXACT: Data transmit in progress
- SDIO_FLAG_RXACT: Data receive in progress
- SDIO_FLAG_TXFIFOHE: Transmit FIFO Half Empty
- SDIO_FLAG_RXFIFOHF: Receive FIFO Half Full
- SDIO_FLAG_TXFIFOF: Transmit FIFO full
- SDIO_FLAG_RXFIFOF: Receive FIFO full
- SDIO_FLAG_TXFIFOE: Transmit FIFO empty
- SDIO_FLAG_RXFIFOE: Receive FIFO empty
- SDIO_FLAG_TXDAVL: Data available in transmit FIFO
- SDIO_FLAG_RXDAVL: Data available in receive FIFO
- SDIO_FLAG_SDIOIT: SD I/O interrupt received

Return value:

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

__HAL_SD_CLEAR_FLAG

Description:

- Clear the SD's pending flags.

Parameters:

- __HANDLE__: SD Handle
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDIO_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDIO_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDIO_FLAG_CTIMEOUT: Command response timeout
  - SDIO_FLAG_DTIMEOUT: Data timeout
  - SDIO_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDIO_FLAG_RXOVERR: Received FIFO overrun error
  - SDIO_FLAG_CMDREND: Command response received (CRC check passed)
  - SDIO_FLAG_CMDSENT: Command sent (no response required)
  - SDIO_FLAG_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
  - SDIO_FLAG_DBCKEND: Data block sent/received (CRC check passed)

– SDIO_FLAG_SDIOIT: SD I/O interrupt received

Return value:

- None

__HAL_SD_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:
  – SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  – SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  – SDIO_IT_CTIMEOUT: Command response timeout interrupt
  – SDIO_IT_DTIMEOUT: Data timeout interrupt
  – SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  – SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
  – SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
  – SDIO_IT_CMDSENT: Command sent (no response required) interrupt
  – SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
  – SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  – SDIO_IT_CMDACT: Command transfer in progress interrupt
  – SDIO_IT_TXACT: Data transmit in progress interrupt
  – SDIO_IT_RXACT: Data receive in progress interrupt
  – SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  – SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  – SDIO_IT_TXFIFOF: Transmit FIFO full interrupt
  – SDIO_IT_RXFIFOF: Receive FIFO full interrupt
  – SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
  – SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
  – SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
  – SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
  – SDIO_IT_SDIOIT: SD I/O interrupt received

interrupt

Return value:

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

__HAL_SD_CLEAR_IT

Description:

- Clear the SD's interrupt pending bits.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO_IT_CTIMEOUT: Command response timeout interrupt
  - SDIO_IT_DTIMEOUT: Data timeout interrupt
  - SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDIO_IT_CMDSENT: Command sent (no response required) interrupt
  - SDIO_IT_DATAEND: Data end (data counter, SDMMC_DCOUNT, is zero) interrupt
  - SDIO_IT_SDIOIT: SD I/O interrupt received interrupt

Return value:

- None

SD Handle Structure definition

SD_InitTypeDef

SD_TypeDef

38 HAL SMARTCARD Generic Driver

38.1 SMARTCARD Firmware driver registers structures

38.1.1 SMARTCARD_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t Prescaler***
- ***uint32_t GuardTime***
- ***uint32_t NACKState***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
  This member configures the SmartCard communication baud rate. The baud rate is computed using the following formula:IntegerDivider = ((PCLKx) / (16 * (hsmartcard->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***SMARTCARD_Word_Length***
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***SMARTCARD_Stop_Bits***
- ***uint32_t SMARTCARD_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t SMARTCARD_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint32_t SMARTCARD_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint32_t SMARTCARD_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***
- ***uint32_t SMARTCARD_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***
- ***uint32_t SMARTCARD_InitTypeDef::Prescaler***
  Specifies the SmartCard Prescaler value used for dividing the system clock to provide

the smartcard clock. The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency. This parameter can be a value of ***SMARTCARD_Prescaler***

- ***uint32_t SMARTCARD_InitTypeDef::GuardTime***
  Specifies the SmartCard Guard Time value in terms of number of baud clocks
- ***uint32_t SMARTCARD_InitTypeDef::NACKState***
  Specifies the SmartCard NACK Transmission state This parameter can be a value of ***SMARTCARD_NACK_State***

38.1.2 SMARTCARD_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters
- ***uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**

- ***__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code

38.2 SMARTCARD Firmware driver API description

38.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure.
2. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   a. Enable the USARTx interface clock.
   b. SMARTCARD pins configuration:
      – Enable the clock for the SMARTCARD GPIOs.
      – Configure the SMARTCARD pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
      – Configure the USARTx interrupt priority.
      – Enable the NVIC USART IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
      – Declare a DMA handle structure for the Tx/Rx channel.
      – Enable the DMAx interface clock.
      – Configure the declared DMA handle structure with the required Tx/Rx parameters.
      – Configure the DMA Tx/Rx channel.
      – Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
      – Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the SMARTCARD Init structure.
4. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   – These APIs configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.



The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver:

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()

- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback

SMARTCARD HAL driver macros list

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

- __HAL_SMARTCARD_ENABLE: Enable the SMARTCARD peripheral
- __HAL_SMARTCARD_DISABLE: Disable the SMARTCARD peripheral
- __HAL_SMARTCARD_GET_FLAG : Check whether the specified SMARTCARD flag is set or not
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt



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

Additional remark: 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. The SMARTCARD frame formats depend on the frame length defined by the M bit (8-bits or 9-bits). Refer to the product reference manual for details.

38.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in Smartcard mode.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- For the Smartcard mode only these parameters can be configured:
  - Baud Rate
  - Word Length => Should be 9 bits (8 bits + parity)
  - Stop Bit
  - Parity: => Should be enabled
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes
  - Prescaler
  - GuardTime
  - NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
  - Word Length = 9 Bits
  - 1.5 Stop Bit
  - Even parity
  - BaudRate = 12096 baud
  - Tx and Rx enabled

Please refer to the ISO 7816-3 specification for more details.

It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

The HAL_SMARTCARD_Init() function follows the USART SmartCard configuration procedure (details for the procedure are available in reference manual (RM0329)).

This section contains the following APIs:

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***
- ***HAL_SMARTCARD_ReInit()***

38.2.3 IO operation functions

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

1. Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.
2. The USART should be configured as:
   - 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
   - 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.
3. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - Non Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected
4. Blocking mode APIs are :
   - HAL_SMARTCARD_Transmit()
   - HAL_SMARTCARD_Receive()
5. Non Blocking mode APIs with Interrupt are :
   - HAL_SMARTCARD_Transmit_IT()
   - HAL_SMARTCARD_Receive_IT()
   - HAL_SMARTCARD_IRQHandler()
6. Non Blocking mode functions with DMA are :
   - HAL_SMARTCARD_Transmit_DMA()
   - HAL_SMARTCARD_Receive_DMA()
7. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_SMARTCARD_TxCpltCallback()
   - HAL_SMARTCARD_RxCpltCallback()
   - HAL_SMARTCARD_ErrorCallback()

This section contains the following APIs:

- ***HAL_SMARTCARD_Transmit()***
- ***HAL_SMARTCARD_Receive()***
- ***HAL_SMARTCARD_Transmit_IT()***
- ***HAL_SMARTCARD_Receive_IT()***
- ***HAL_SMARTCARD_Transmit_DMA()***
- ***HAL_SMARTCARD_Receive_DMA()***
- ***HAL_SMARTCARD_Abort()***
- ***HAL_SMARTCARD_AbortTransmit()***
- ***HAL_SMARTCARD_AbortReceive()***
- ***HAL_SMARTCARD_Abort_IT()***
- ***HAL_SMARTCARD_AbortTransmit_IT()***
- ***HAL_SMARTCARD_AbortReceive_IT()***
- ***HAL_SMARTCARD_IRQHandler()***
- ***HAL_SMARTCARD_TxCpltCallback()***

- ***HAL_SMARTCARD_RxCpltCallback()***
- ***HAL_SMARTCARD_ErrorCallback()***
- ***HAL_SMARTCARD_AbortCpltCallback()***
- ***HAL_SMARTCARD_AbortTransmitCpltCallback()***
- ***HAL_SMARTCARD_AbortReceiveCpltCallback()***

38.2.4 Peripheral State and Errors functions

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

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SmartCard peripheral.
- HAL_SMARTCARD_GetError() check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- ***HAL_SMARTCARD_GetState()***
- ***HAL_SMARTCARD_GetError()***

38.2.5 Detailed description of functions

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	• **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARD_ReInit

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_ReInit (SMARTCARD_HandleTypeDef * hsc)**
Function description	

HAL_SMARTCARD_DeInit

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsc)**
Function description	DeInitializes the USART SmartCard peripheral.
Parameters	• **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
Return values	• **HAL:** status

HAL_SMARTCARD_MspInit

Function name **void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)**

Function description SMARTCARD MSP Init.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **None:**

HAL_SMARTCARD_MspDeInit

Function name **void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsc)**

Function description SMARTCARD MSP DeInit.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **None:**

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
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

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
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** Timeout duration

Return values
- **HAL:** status

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 non blocking mode.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

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 non blocking mode.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be received

Return values
- **HAL:** status

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 non blocking mode.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

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 non blocking mode.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** pointer to data buffer

- **Size:** amount of data to be received

Return values
- **HAL:** status

Notes
- When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bit.s

HAL_SMARTCARD_Abort

Function name **HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsc)**

Function description Abort ongoing transfers (blocking mode).

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name **HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsc)**

Function description Abort ongoing Transmit transfer (blocking mode).

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name **HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsc)**

Function description Abort ongoing Receive transfer (blocking mode).

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable

the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsc)**
Function description	Abort ongoing transfers (Interrupt mode).
Parameters	• **hsc:** SMARTCARD handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsc)**
Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **hsc:** SMARTCARD handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsc)**
Function description	Abort ongoing Receive transfer (Interrupt mode).

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name **void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)**

Function description This function handles SMARTCARD interrupt request.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **None:**

HAL_SMARTCARD_TxCpltCallback

Function name **void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function description Tx Transfer completed callbacks.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **None:**

HAL_SMARTCARD_RxCpltCallback

Function name **void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function description Rx Transfer completed callbacks.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **None:**

HAL_SMARTCARD_ErrorCallback

Function name **void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsc)**

Function description SMARTCARD error callbacks.

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name **void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function description SMARTCARD Abort Complete callback.

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name **void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function description SMARTCARD Abort Transmit Complete callback.

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name **void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function description SMARTCARD Abort ReceiveComplete callback.

Parameters
- **hsc:** SMARTCARD handle.

Return values
- **None:**

HAL_SMARTCARD_GetState

Function name **HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsc)**

Function description return the SMARTCARD state

Parameters
- **hsc:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

Return values
- **HAL:** state

HAL_SMARTCARD_GetError

Function name **uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsc)**

Function description Return the SMARTCARD error code.

Parameters
- **hsc:** : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.

Return values
- **SMARTCARD:** Error Code

38.3 SMARTCARD Firmware driver defines

38.3.1 SMARTCARD

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD_PHASE_2EDGE

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD_POLARITY_HIGH

SMARTCARD DMA requests

SMARTCARD_DMAREQ_TX

SMARTCARD_DMAREQ_RX

SMARTCARD Error Code

HAL_SMARTCARD_ERROR_NONE	No error
HAL_SMARTCARD_ERROR_PE	Parity error
HAL_SMARTCARD_ERROR_NE	Noise error
HAL_SMARTCARD_ERROR_FE	Frame error
HAL_SMARTCARD_ERROR_ORE	OverRun error
HAL_SMARTCARD_ERROR_DMA	DMA transfer error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:
- Reset SMARTCARD handle gstate & RxState.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:
- Flush the Smartcard DR register.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the

USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_TXE: Transmit data register empty flag
  - SMARTCARD_FLAG_TC: Transmission Complete flag
  - SMARTCARD_FLAG_RXNE: Receive data register not empty flag
  - SMARTCARD_FLAG_IDLE: Idle Line detection flag
  - SMARTCARD_FLAG_ORE: OverRun Error flag
  - SMARTCARD_FLAG_NE: Noise Error flag
  - SMARTCARD_FLAG_FE: Framing Error flag
  - SMARTCARD_FLAG_PE: Parity Error flag

Return value:

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

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified Smartcard pending flags.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any

combination of the following values:
- SMARTCARD_FLAG_TC: Transmission Complete flag.
- SMARTCARD_FLAG_RXNE: Receive data register not empty flag.

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error) and ORE (OverRun error) flags are cleared by software sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value

depending on device).

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_TXE: Transmit Data Register empty interrupt
  - SMARTCARD_IT_TC: Transmission complete interrupt
  - SMARTCARD_IT_RXNE: Receive Data register not empty interrupt
  - SMARTCARD_IT_IDLE: Idle line detection interrupt
  - SMARTCARD_IT_PE: Parity Error interrupt
  - SMARTCARD_IT_ERR: Error interrupt(Frame error, noise

error, overRun error)

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_TXE: Transmit Data Register empty interrupt
  - SMARTCARD_IT_TC: Transmission complete interrupt
  - SMARTCARD_IT_RXNE: Receive Data register not empty interrupt
  - SMARTCARD_IT_IDLE: Idle line detection interrupt
  - SMARTCARD_IT_PE: Parity Error interrupt
  - SMARTCARD_IT_ERR: Error interrupt(Frame error, noise error, overRun error)

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the SmartCard Handle.
- __IT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_TXE: Transmit Data Register empty interrupt
  - SMARTCARD_IT_TC: Transmission complete interrupt
  - SMARTCARD_IT_RXNE: Receive Data register not

empty interrupt
- SMARTCARD_IT_IDLE: Idle line detection interrupt
- SMARTCARD_IT_ERR: Error interrupt
- SMARTCARD_IT_PE: Parity Error interrupt

Return value:

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

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_SMARTCARD_DMA_REQUEST_ENABLE

Description:

- Macros to enable the SmartCard DMA request.

Parameters:

- __HANDLE__: specifies the SmartCard Handle.
- __REQUEST__: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - SMARTCARD_DMAREQ_TX: SmartCard DMA transmit request
  - SMARTCARD_DMAREQ_RX: SmartCard DMA receive request

__HAL_SMARTCARD_DMA_REQUEST_DISABLE	**Description:** • Macros to disable the SmartCard DMA request. **Parameters:** • __HANDLE__: specifies the SmartCard Handle. • __REQUEST__: specifies the SmartCard DMA request. This parameter can be one of the following values: – SMARTCARD_DMAREQ_TX: SmartCard DMA transmit request – SMARTCARD_DMAREQ_RX: SmartCard DMA receive request

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD_LASTBIT_ENABLE

SMARTCARD Mode

SMARTCARD_MODE_RX

SMARTCARD_MODE_TX

SMARTCARD_MODE_TX_RX

SMARTCARD NACK State

SMARTCARD_NACK_ENABLE

SMARTCARD_NACK_DISABLE

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD_PARITY_ODD

SMARTCARD Prescaler

SMARTCARD_PRESCALER_SYSCLK_DIV2	SYSCLK divided by 2
SMARTCARD_PRESCALER_SYSCLK_DIV4	SYSCLK divided by 4
SMARTCARD_PRESCALER_SYSCLK_DIV6	SYSCLK divided by 6
SMARTCARD_PRESCALER_SYSCLK_DIV8	SYSCLK divided by 8
SMARTCARD_PRESCALER_SYSCLK_DIV10	SYSCLK divided by 10
SMARTCARD_PRESCALER_SYSCLK_DIV12	SYSCLK divided by 12
SMARTCARD_PRESCALER_SYSCLK_DIV14	SYSCLK divided by 14
SMARTCARD_PRESCALER_SYSCLK_DIV16	SYSCLK divided by 16
SMARTCARD_PRESCALER_SYSCLK_DIV18	SYSCLK divided by 18
SMARTCARD_PRESCALER_SYSCLK_DIV20	SYSCLK divided by 20

SMARTCARD_PRESCALER_SYSCLK_DIV22	SYSCLK divided by 22
SMARTCARD_PRESCALER_SYSCLK_DIV24	SYSCLK divided by 24
SMARTCARD_PRESCALER_SYSCLK_DIV26	SYSCLK divided by 26
SMARTCARD_PRESCALER_SYSCLK_DIV28	SYSCLK divided by 28
SMARTCARD_PRESCALER_SYSCLK_DIV30	SYSCLK divided by 30
SMARTCARD_PRESCALER_SYSCLK_DIV32	SYSCLK divided by 32
SMARTCARD_PRESCALER_SYSCLK_DIV34	SYSCLK divided by 34
SMARTCARD_PRESCALER_SYSCLK_DIV36	SYSCLK divided by 36
SMARTCARD_PRESCALER_SYSCLK_DIV38	SYSCLK divided by 38
SMARTCARD_PRESCALER_SYSCLK_DIV40	SYSCLK divided by 40
SMARTCARD_PRESCALER_SYSCLK_DIV42	SYSCLK divided by 42
SMARTCARD_PRESCALER_SYSCLK_DIV44	SYSCLK divided by 44
SMARTCARD_PRESCALER_SYSCLK_DIV46	SYSCLK divided by 46
SMARTCARD_PRESCALER_SYSCLK_DIV48	SYSCLK divided by 48
SMARTCARD_PRESCALER_SYSCLK_DIV50	SYSCLK divided by 50
SMARTCARD_PRESCALER_SYSCLK_DIV52	SYSCLK divided by 52
SMARTCARD_PRESCALER_SYSCLK_DIV54	SYSCLK divided by 54
SMARTCARD_PRESCALER_SYSCLK_DIV56	SYSCLK divided by 56
SMARTCARD_PRESCALER_SYSCLK_DIV58	SYSCLK divided by 58
SMARTCARD_PRESCALER_SYSCLK_DIV60	SYSCLK divided by 60
SMARTCARD_PRESCALER_SYSCLK_DIV62	SYSCLK divided by 62

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD_STOPBITS_1_5

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

39 HAL SPI Generic Driver

39.1 SPI Firmware driver registers structures

39.1.1 SPI_InitTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI Directional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note: The communication clock is derived from the master clock. The slave clock does not need to be set.
- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***

- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0 and Max_Data = 65535

39.1.2 __SPI_HandleTypeDef

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state

- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code

39.2 SPI Firmware driver API description

39.2.1 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 initilalized hdma_tx(or _rx) 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 customized HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
   a. Master 2Lines RxOnly
   b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

Master Receive mode restriction:

1. In Master unidirectional receive-only mode (MSTR =1, BIDIMODE=0, RXONLY=0) or bidirectional receive mode (MSTR=1, BIDIMODE=1, BIDIOE=0), to ensure that the SPI does not initiate a new transfer the following procedure has to be respected:
   a. HAL_SPI_DeInit()
   b. HAL_SPI_Init()

39.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***

39.2.3 IO operation functions

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

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_Abort()***

- ***HAL_SPI_Abort_IT()***
- ***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()***
- ***HAL_SPI_AbortCpltCallback()***

39.2.4 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()***

39.2.5 Detailed description of functions

HAL_SPI_Init

Function name	**HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)**
Function description	Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL:** status

HAL_SPI_DeInit

Function name	**HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)**
Function description	De Initialize the SPI peripheral.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL:** status

HAL_SPI_MspInit

Function name	**void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)**
Function description	Initialize the SPI MSP.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **None:**

HAL_SPI_MspDeInit

Function name **void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)**

Function description De-Initialize the SPI MSP.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **None:**

HAL_SPI_Transmit

Function name **HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Transmit an amount of data in blocking mode.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_SPI_Receive

Function name **HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Receive an amount of data in blocking mode.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_SPI_TransmitReceive

Function name **HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**

Function description Transmit and Receive an amount of data in blocking mode.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received

- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_SPI_Transmit_IT

Function name **HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**

Function description Transmit an amount of data in non-blocking mode with Interrupt.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

HAL_SPI_Receive_IT

Function name **HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**

Function description Receive an amount of data in non-blocking mode with Interrupt.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name **HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**

Function description Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received

Return values
- **HAL:** status

HAL_SPI_Transmit_DMA

Function name **HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**

Function description Transmit an amount of data in non-blocking mode with DMA.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer

- **Size:** amount of data to be sent

Return values
- **HAL:** status

HAL_SPI_Receive_DMA

Function name **HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**

Function description Receive an amount of data in non-blocking mode with DMA.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

Notes
- When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name **HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**

Function description Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent

Return values
- **HAL:** status

Notes
- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name **HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)**

Function description Pause the DMA Transfer.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values
- **HAL:** status

HAL_SPI_DMAResume

Function name **HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)**

Function description Resume the DMA Transfer.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values
- **HAL:** status

HAL_SPI_DMAStop

Function name **HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)**

Function description Stop the DMA Transfer.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values
- **HAL:** status

HAL_SPI_Abort

Function name **HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)**

Function description Abort ongoing transfer (blocking mode).

Parameters
- **hspi:** SPI handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode. This procedure performs following operations : Disable SPI Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.
- Once transfer is aborted, the __HAL_SPI_CLEAR_OVRFLAG() macro must be called in user application before starting new SPI receive process.

HAL_SPI_Abort_IT

Function name **HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)**

Function description Abort ongoing transfer (Interrupt mode).

Parameters
- **hspi:** SPI handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode. This procedure performs following operations : Disable SPI Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback

- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).
- Once transfer is aborted, the __HAL_SPI_CLEAR_OVRFLAG() macro must be called in user application before starting new SPI receive process.

HAL_SPI_IRQHandler

Function name	**void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)**
Function description	Handle SPI interrupt request.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	• **None:**

HAL_SPI_TxCpltCallback

Function name	**void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_RxCpltCallback

Function name	**void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Rx Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_TxRxCpltCallback

Function name	**void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx and Rx Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **None:**

HAL_SPI_TxHalfCpltCallback

Function name	**void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function description	Tx Half Transfer completed callback.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **None:**

HAL_SPI_RxHalfCpltCallback

Function name **void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)**

Function description Rx Half Transfer completed callback.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name **void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)**

Function description Tx and Rx Half Transfer callback.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **None:**

HAL_SPI_ErrorCallback

Function name **void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)**

Function description SPI error callback.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **None:**

HAL_SPI_AbortCpltCallback

Function name **void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)**

Function description SPI Abort Complete callback.

Parameters
- **hspi:** SPI handle.

Return values
- **None:**

HAL_SPI_GetState

Function name **HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)**

Function description Return the SPI handle state.

Parameters
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values
- **SPI:** state

HAL_SPI_GetError

Function name	**uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)**
Function description	Return the SPI error code.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **SPI:** error code in bitmap format

SPI_ISCRCErrorValid

Function name	**uint8_t SPI_ISCRCErrorValid (SPI_HandleTypeDef * hspi)**
Function description	Checks if encountered CRC error could be corresponding to wrongly detected errors according to SPI instance, Device type, and revision ID.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **CRC:** error validity (SPI_INVALID_CRC_ERROR or SPI_VALID_CRC_ERROR).

39.3 SPI Firmware driver defines

39.3.1 SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI Data Size

SPI_DATASIZE_8BIT

SPI_DATASIZE_16BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Code

HAL_SPI_ERROR_NONE	No error
HAL_SPI_ERROR_MODF	MODF error
HAL_SPI_ERROR_CRC	CRC error
HAL_SPI_ERROR_OVR	OVR error
HAL_SPI_ERROR_FRE	FRE error
HAL_SPI_ERROR_DMA	DMA transfer error
HAL_SPI_ERROR_FLAG	Flag: RXNE,TXE, BSY

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

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

__HAL_SPI_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXNE: Receive buffer not empty flag
  - SPI_FLAG_TXE: Transmit buffer empty flag
  - SPI_FLAG_CRCERR: CRC error flag

– SPI_FLAG_MODF: Mode fault flag
– SPI_FLAG_OVR: Overrun flag
– SPI_FLAG_BSY: Busy flag

Return value:

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

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

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:

- Disable the SPI peripheral.

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 Flags Definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_BSY

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI Interrupt 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 Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

40 HAL SRAM Generic Driver

40.1 SRAM Firmware driver registers structures

40.1.1 SRAM_HandleTypeDef

Data Fields

- ***FSMC_NORSRAM_TypeDef * Instance***
- ***FSMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FSMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***

Field Documentation

- ***FSMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance***
  Register base address
- ***FSMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FSMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* SRAM_HandleTypeDef::hdma***
  Pointer DMA handler

40.2 SRAM Firmware driver API description

40.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FSMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FSMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example:
   SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FSMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FSMC_NORSRAM_TimingTypeDef Timing and FSMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()

b. Control register configuration using the FSMC NORSRAM interface function FSMC_NORSRAM_Init()
c. Timing register configuration using the FSMC NORSRAM interface function FSMC_NORSRAM_Timing_Init()
d. Extended mode Timing register configuration using the FSMC NORSRAM interface function FSMC_NORSRAM_Extended_Timing_Init()
e. Enable the SRAM device using the macro __FSMC_NORSRAM_ENABLE()

4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   – HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

40.2.2 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()***

40.2.3 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()***

40.2.4 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()***

40.2.5 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()***

40.2.6 Detailed description of functions

HAL_SRAM_Init

Function name	**HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)**
Function description	Performs the SRAM device initialization sequence.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **Timing:** Pointer to SRAM control timing structure • **ExtTiming:** Pointer to SRAM extended mode timing structure
Return values	• **HAL:** status

HAL_SRAM_DeInit

Function name	**HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)**
Function description	Performs the SRAM device De-initialization sequence.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **HAL:** status

HAL_SRAM_MspInit

Function name	**void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)**
Function description	SRAM MSP Init.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **None:**

HAL_SRAM_MspDeInit

Function name	**void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)**
Function description	SRAM MSP DeInit.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• **None:**

HAL_SRAM_DMA_XferCpltCallback

Function name **void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)**

Function description DMA transfer complete callback.

Parameters
- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values
- **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name **void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)**

Function description DMA transfer complete error callback.

Parameters
- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values
- **None:**

HAL_SRAM_Read_8b

Function name **HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)**

Function description Reads 8-bit buffer from SRAM memory.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values
- **HAL:** status

HAL_SRAM_Write_8b

Function name **HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)**

Function description Writes 8-bit buffer to SRAM memory.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values
- **HAL:** status

HAL_SRAM_Read_16b

Function name **HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t**

*** pDstBuffer, uint32_t BufferSize)**

Function description: Reads 16-bit buffer from SRAM memory.

Parameters:
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values:
- **HAL:** status

HAL_SRAM_Write_16b

Function name: **HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)**

Function description: Writes 16-bit buffer to SRAM memory.

Parameters:
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values:
- **HAL:** status

HAL_SRAM_Read_32b

Function name: **HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)**

Function description: Reads 32-bit buffer from SRAM memory.

Parameters:
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values:
- **HAL:** status

HAL_SRAM_Write_32b

Function name: **HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)**

Function description: Writes 32-bit buffer to SRAM memory.

Parameters:
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values:
- **HAL:** status

HAL_SRAM_Read_DMA

Function name **HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)**

Function description Reads a Words data from the SRAM memory using DMA transfer.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values
- **HAL:** status

HAL_SRAM_Write_DMA

Function name **HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)**

Function description Writes a Words data buffer to SRAM memory using DMA transfer.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values
- **HAL:** status

HAL_SRAM_WriteOperation_Enable

Function name **HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)**

Function description Enables dynamically SRAM write operation.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values
- **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name **HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)**

Function description Disables dynamically SRAM write operation.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values
- **HAL:** status

HAL_SRAM_GetState

Function name **HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)**

Function description Returns the SRAM controller state.

Parameters
- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values
- **HAL:** state

40.3 SRAM Firmware driver defines

40.3.1 SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:
- Reset SRAM handle state.

Parameters:
- __HANDLE__: SRAM handle

Return value:
- None

41 HAL TIM Generic Driver

41.1 TIM Firmware driver registers structures

41.1.1 TIM_Base_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:the number of PWM periods in edge-aligned modethe number of half PWM period in center-aligned mode This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***
  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

41.1.2 TIM_OC_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
  Note: This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.

41.1.3 TIM_OnePulse_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note: This parameter is valid only for TIM1 and TIM8.

- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.1.4 TIM_IC_InitTypeDef

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.1.5 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

41.1.6 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

41.1.7 TIM_ClearInputConfigTypeDef

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter can be a value of ***TIM_ClearInput_Prescaler***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.1.8 TIM_SlaveConfigTypeDef

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.1.9 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[7U]***
  DMA Handlers array This array is accessed by a ***TIM_DMA_Handle_index***
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
  TIM operation state

41.2 TIM Firmware driver API description

41.2.1 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

41.2.2 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 __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      – Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_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()

41.2.3 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()***

41.2.4 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()***

41.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- 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()***

41.2.6 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()***

41.2.7 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()***

41.2.8 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()***

41.2.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

- ***HAL_TIM_IRQHandler()***

41.2.10 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()***

41.2.11 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()***

41.2.12 Peripheral State 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_TIM_Base_GetState()*
- *HAL_TIM_OC_GetState()*
- *HAL_TIM_PWM_GetState()*
- *HAL_TIM_IC_GetState()*
- *HAL_TIM_OnePulse_GetState()*
- *HAL_TIM_Encoder_GetState()*

41.2.13 Detailed description of functions

TIM_Base_SetConfig

Function name **void TIM_Base_SetConfig (TIM_TypeDef * TIMx, TIM_Base_InitTypeDef * Structure)**

Function description Time Base configuration.

Parameters
- **TIMx:** : TIM periheral
- **Structure:** : TIM Base configuration structure

Return values
- **None:**

TIM_TI1_SetConfig

Function name **void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)**

Function description Configure the TI1 as Input.

Parameters
- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** : The Input Polarity. This parameter can be one of the following values:
  - TIM_ICPOLARITY_RISING
  - TIM_ICPOLARITY_FALLING
- **TIM_ICSelection:** : specifies the input to be used. This parameter can be one of the following values:
  - TIM_ICSELECTION_DIRECTTI: TIM Input 1 is selected to be connected to IC1.
  - TIM_ICSELECTION_INDIRECTTI: TIM Input 1 is selected to be connected to IC2.
  - TIM_ICSELECTION_TRC: TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** : Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values
- **None:**

Notes
- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name **void TIM_OC2_SetConfig (TIM_TypeDef * TIMx,**

TIM_OC_InitTypeDef * OC_Config)

Function description: Time Ouput Compare 2 configuration.

Parameters:
- **TIMx:** to select the TIM peripheral
- **OC_Config:** : The ouput configuration structure

Return values:
- **None:**

TIM_DMADelayPulseCplt

Function name: **void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)**

Function description: TIM DMA Delay Pulse complete callback.

Parameters:
- **hdma:** : pointer to DMA handle.

Return values:
- **None:**

TIM_DMAError

Function name: **void TIM_DMAError (DMA_HandleTypeDef * hdma)**

Function description: TIM DMA error callback.

Parameters:
- **hdma:** : pointer to DMA handle.

Return values:
- **None:**

TIM_DMACaptureCplt

Function name: **void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)**

Function description: TIM DMA Capture complete callback.

Parameters:
- **hdma:** : pointer to DMA handle.

Return values:
- **None:**

TIM_CCxChannelCmd

Function name: **void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)**

Function description: Enables or disables the TIM Capture Compare Channel x.

Parameters:
- **TIMx:** to select the TIM peripheral
- **Channel:** : specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
- **ChannelState:** : specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_Disable.

Return values:
- **None:**

HAL_TIM_Base_Init

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM Base handle
Return values	• **HAL:** status

HAL_TIM_Base_DeInit

Function name	**HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes the TIM Base peripheral.
Parameters	• **htim:** : TIM Base handle
Return values	• **HAL:** status

HAL_TIM_Base_MspInit

Function name	**void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)**
Function description	Initializes the TIM Base MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_Base_MspDeInit

Function name	**void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes TIM Base MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_Base_Start

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Base generation.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIM_Base_Stop

Function name	**HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Base generation.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIM_Base_Start_IT

Function name **HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)**

Function description Starts the TIM Base generation in interrupt mode.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIM_Base_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)**

Function description Stops the TIM Base generation in interrupt mode.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIM_Base_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM Base generation in DMA mode.

Parameters
- **htim:** : TIM handle
- **pData:** : The source Buffer address.
- **Length:** : The length of data to be transferred from memory to peripheral.

Return values
- **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)**

Function description Stops the TIM Base generation in DMA mode.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIM_OC_Init

Function name **HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters
- **htim:** : TIM Output Compare handle

Return values
- **HAL:** status

HAL_TIM_OC_DeInit

Function name **HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes the TIM peripheral.

Parameters
- **htim:** : TIM Output Compare handle

Return values
- **HAL:** status

HAL_TIM_OC_MspInit

Function name **void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM Output Compare MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_OC_MspDeInit

Function name **void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes TIM Output Compare MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_OC_Start

Function name **HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Output Compare signal generation.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_OC_Stop

Function name **HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Output Compare signal generation.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can

be one of the following values:
- TIM_CHANNEL_1: TIM Channel 1 selected
- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_OC_Start_IT

Function name **HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Output Compare signal generation in interrupt mode.

Parameters
- **htim:** : TIM OC handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Output Compare signal generation in interrupt mode.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM Output Compare signal generation in DMA mode.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

– TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** : The source Buffer address.
- **Length:** : The length of data to be transferred from memory to TIM peripheral

Return values
- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Output Compare signal generation in DMA mode.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_PWM_Init

Function name **HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIM_PWM_DeInit

Function name **HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes the TIM peripheral.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIM_PWM_MspInit

Function name **void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM PWM MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_PWM_MspDeInit

Function name **void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes TIM PWM MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_PWM_Start

Function name **HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the PWM signal generation.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_PWM_Stop

Function name **HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the PWM signal generation.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_PWM_Start_IT

Function name **HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the PWM signal generation in interrupt mode.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name
HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description
Stops the PWM signal generation in interrupt mode.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name
HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description
Starts the TIM PWM signal generation in DMA mode.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** : The source Buffer address.
- **Length:** : The length of data to be transferred from memory to TIM peripheral

Return values
- **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name
HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description
Stops the TIM PWM signal generation in DMA mode.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_Init

Function name **HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters
- **htim:** : TIM Input Capture handle

Return values
- **HAL:** status

HAL_TIM_IC_DeInit

Function name **HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes the TIM peripheral.

Parameters
- **htim:** : TIM Input Capture handle

Return values
- **HAL:** status

HAL_TIM_IC_MspInit

Function name **void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM Input Capture MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_IC_MspDeInit

Function name **void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes TIM Input Capture MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_IC_Start

Function name **HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Input Capture measurement.

Parameters
- **htim:** : TIM Input Capture handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_Stop

Function name **HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Input Capture measurement.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_Start_IT

Function name **HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Input Capture measurement in interrupt mode.

Parameters
- **htim:** : TIM Input Capture handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Input Capture measurement in interrupt mode.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM Input Capture measurement in DMA mode.

Parameters
- **htim:** : TIM Input Capture handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** : The destination Buffer address.
- **Length:** : The length of data to be transferred from TIM peripheral to memory.

Return values
- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Input Capture measurement in DMA mode.

Parameters
- **htim:** : TIM Input Capture handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_OnePulse_Init

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)**

Function description Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters
- **htim:** : TIM OnePulse handle
- **OnePulseMode:** : Select the One pulse mode. This parameter can be one of the following values:
  - TIM_OPMODE_SINGLE: Only one pulse will be generated.
  - TIM_OPMODE_REPETITIVE: Repetitive pulses wil be generated.

Return values
- **HAL:** status

HAL_TIM_OnePulse_DeInit

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes the TIM One Pulse.

Parameters
- **htim:** : TIM One Pulse handle

Return values
- **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name **void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM One Pulse MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_OnePulse_MspDeInit

Function name **void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes TIM One Pulse MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_OnePulse_Start

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Starts the TIM One Pulse signal generation.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIM_OnePulse_Stop

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Stops the TIM One Pulse signal generation.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channels to be disable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIM_OnePulse_Start_IT

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Starts the TIM One Pulse signal generation in interrupt mode.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIM_OnePulse_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Stops the TIM One Pulse signal generation in interrupt mode.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIM_Encoder_Init

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)**

Function description Initializes the TIM Encoder Interface and create the associated handle.

Parameters
- **htim:** : TIM Encoder Interface handle
- **sConfig:** : TIM Encoder Interface configuration structure

Return values
- **HAL:** status

HAL_TIM_Encoder_DeInit

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes the TIM Encoder interface.

Parameters
- **htim:** : TIM Encoder handle

Return values
- **HAL:** status

HAL_TIM_Encoder_MspInit

Function name **void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)**

Function description Initializes the TIM Encoder Interface MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_Encoder_MspDeInit

Function name **void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)**

Function description DeInitializes TIM Encoder Interface MSP.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_Encoder_Start

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Encoder Interface.

Parameters
- **htim:** : TIM Encoder Interface handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values
- **HAL:** status

HAL_TIM_Encoder_Stop

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Encoder Interface.

Parameters
- **htim:** : TIM Encoder Interface handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values
- **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Encoder Interface in interrupt mode.

Parameters
- **htim:** : TIM Encoder Interface handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values
- **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Encoder Interface in interrupt mode.

Parameters
- **htim:** : TIM Encoder Interface handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values
- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)**

Function description Starts the TIM Encoder Interface in DMA mode.

Parameters
- **htim:** : TIM Encoder Interface handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_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

HAL_TIM_Encoder_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Encoder Interface in DMA mode.

Parameters
- **htim:** : TIM Encoder Interface handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values
- **HAL:** status

HAL_TIM_IRQHandler

Function name **void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)**

Function description This function handles TIM interrupts requests.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIM_OC_ConfigChannel

Function name **HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**

Function description Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters
- **htim:** : TIM Output Compare handle
- **sConfig:** : TIM Output Compare configuration structure
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name **HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**

Function description Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters
- **htim:** : TIM handle
- **sConfig:** : TIM PWM configuration structure
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name **HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)**

Function description Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters
- **htim:** : TIM IC handle
- **sConfig:** : TIM Input Capture configuration structure
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name **HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)**

Function description Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters
- **htim:** : TIM One Pulse handle
- **sConfig:** : TIM One Pulse configuration structure
- **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** : TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIM_ConfigOCrefClear

Function name **HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)**

Function description Configures the OCRef clear feature.

Parameters
- **htim:** : TIM handle
- **sClearInputConfig:** : pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** : specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4

Return values
- **HAL:** status

HAL_TIM_ConfigClockSource

Function name **HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)**

Function description Configures the clock source to be used.

Parameters
- **htim:** : TIM handle
- **sClockSourceConfig:** : pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values
- **HAL:** status

HAL_TIM_ConfigTI1Input

Function name **HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)**

Function description Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters
- **htim:** : TIM handle.
- **TI1_Selection:** : Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
  - TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input
  - TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values
- **HAL:** status

HAL_TIM_SlaveConfigSynchronization

Function name **HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**

Function description Configures the TIM in Slave mode.

Parameters
- **htim:** : TIM handle.
- **sSlaveConfig:** : pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values
- **HAL:** status

HAL_TIM_SlaveConfigSynchronization_IT

Function name **HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef ***

sSlaveConfig)

Function description: Configures the TIM in Slave mode in interrupt mode.

Parameters:
- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values:
- **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name: **HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**

Function description: Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters:
- **htim:** : TIM handle
- **BurstBaseAddress:** : TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_DCR
- **BurstRequestSrc:** : TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** : The Buffer address.

- **BurstLength:** : DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.

Return values
- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name **HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**

Function description Stops the TIM DMA Burst mode.

Parameters
- **htim:** : TIM handle
- **BurstRequestSrc:** : TIM DMA Request sources to disable

Return values
- **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name **HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**

Function description Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters
- **htim:** : TIM handle
- **BurstBaseAddress:** : TIM Base address from where the DMA will starts the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_DCR
- **BurstRequestSrc:** : TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source

  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** : The Buffer address.
- **BurstLength:** : DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.

Return values
- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name **HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**

Function description Stop the DMA burst reading.

Parameters
- **htim:** : TIM handle
- **BurstRequestSrc:** : TIM DMA Request sources to disable.

Return values
- **HAL:** status

HAL_TIM_GenerateEvent

Function name **HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)**

Function description Generate a software event.

Parameters
- **htim:** : TIM handle
- **EventSource:** : specifies the event source. This parameter can be one of the following values:
  - TIM_EVENTSOURCE_UPDATE: Timer update Event source
  - TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  - TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source
  - TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
  - TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
  - TIM_EVENTSOURCE_COM: Timer COM event source
  - TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
  - TIM_EVENTSOURCE_BREAK: Timer Break event source

Return values
- **HAL:** status

Notes
- TIM6 and TIM7 can only generate an update event.
- TIM_EVENTSOURCE_COM and TIM_EVENTSOURCE_BREAK are used only with TIM1, TIM15, TIM16 and TIM17.

HAL_TIM_ReadCapturedValue

Function name	**uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function description	Read the captured value from Capture Compare unit.
Parameters	• **htim:** : TIM handle. • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: – TIM_CHANNEL_1 : TIM Channel 1 selected – TIM_CHANNEL_2 : TIM Channel 2 selected – TIM_CHANNEL_3 : TIM Channel 3 selected – TIM_CHANNEL_4 : TIM Channel 4 selected
Return values	• **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name	**void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)**
Function description	Period elapsed callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name	**void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)**
Function description	Output Compare callback in non blocking mode.
Parameters	• **htim:** : TIM OC handle
Return values	• **None:**

HAL_TIM_IC_CaptureCallback

Function name	**void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)**
Function description	Input Capture callback in non blocking mode.
Parameters	• **htim:** : TIM IC handle
Return values	• **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name	**void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)**
Function description	PWM Pulse finished callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_TriggerCallback

Function name	**void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)**
Function description	Hall Trigger detection callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_ErrorCallback

Function name	**void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)**
Function description	Timer error callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIM_Base_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Base state.
Parameters	• **htim:** : TIM Base handle
Return values	• **HAL:** state

HAL_TIM_OC_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM OC state.
Parameters	• **htim:** : TIM Ouput Compare handle
Return values	• **HAL:** state

HAL_TIM_PWM_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM PWM state.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** state

HAL_TIM_IC_GetState

Function name	**HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)**
Function description	Return the TIM Input Capture state.
Parameters	• **htim:** : TIM IC handle

Return values
- **HAL:** state

HAL_TIM_OnePulse_GetState

Function name **HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)**

Function description Return the TIM One Pulse Mode state.

Parameters
- **htim:** : TIM OPM handle

Return values
- **HAL:** state

HAL_TIM_Encoder_GetState

Function name **HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)**

Function description Return the TIM Encoder Mode state.

Parameters
- **htim:** : TIM Encoder handle

Return values
- **HAL:** state

41.3 TIM Firmware driver defines

41.3.1 TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE TIMx_ARR register is buffered

TIM Break Input Enable Disable

TIM_BREAK_ENABLE

TIM_BREAK_DISABLE

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

TIM_BREAKPOLARITY_HIGH

TIM Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_ALL

TIM Capture/Compare Channel State

TIM_CCx_ENABLE

TIM_CCx_DISABLE

TIM_CCxN_ENABLE

TIM_CCxN_DISABLE

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 ClearInput Source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_NONE

TIM ClockDivision

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 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 Handle Index

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

TIM DMA Sources

TIM_DMA_UPDATE

TIM_DMA_CC1

TIM_DMA_CC2

TIM_DMA_CC3

TIM_DMA_CC4

TIM_DMA_COM

TIM_DMA_TRIGGER

TIM Encoder Mode

TIM_ENCODERMODE_TI1

TIM_ENCODERMODE_TI2

TIM_ENCODERMODE_TI12

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED	Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED	Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1	No prescaler is used
TIM_ETRPRESCALER_DIV2	ETR input source is divided by 2
TIM_ETRPRESCALER_DIV4	ETR input source is divided by 4
TIM_ETRPRESCALER_DIV8	ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

TIM_EVENTSOURCE_CC1

TIM_EVENTSOURCE_CC2

TIM_EVENTSOURCE_CC3

TIM_EVENTSOURCE_CC4

TIM_EVENTSOURCE_COM

TIM_EVENTSOURCE_TRIGGER

TIM_EVENTSOURCE_BREAK

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:

- Enables the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.

- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_DISABLE_IT

Description:

- Disables the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enables the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1

DMA request
- TIM_DMA_CC2: Capture/Compare 2 DMA request
- TIM_DMA_CC3: Capture/Compare 3 DMA request
- TIM_DMA_CC4: Capture/Compare 4 DMA request
- TIM_DMA_COM: Commutation DMA request
- TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:

- Disables the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag

- TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
- TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
- TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
- TIM_FLAG_COM: Commutation interrupt flag
- TIM_FLAG_TRIGGER: Trigger interrupt flag
- TIM_FLAG_BREAK: Break interrupt flag
- TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
- TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
- TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
- TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

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

__HAL_TIM_CLEAR_FLAG

Description:

- Clears the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF:

Capture/Compare 3 overcapture flag
– TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

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

__HAL_TIM_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check.

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear.

Return value:

- None

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly usefull to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:

- Sets the TIM active prescaler register value on update event.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the active prescaler

register new value.

Return value:

- None

__HAL_TIM_SET_COMPARE

Description:

- Sets the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: : TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE

Description:

- Gets the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: : TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value
  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value

Return value:

- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__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:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__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:

- @retval: 16-bit or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the

following value:
- TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
- TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
- TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__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:

- The: input capture prescaler 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

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): (+) Counter overflow/underflow (+) Setting the UG bit (+) Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Sets the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

Notes:

- The polarity TIM_INPUTCHANNELPOLARITY_BOTHEDGE is not authorized for TIM Channel 4.

TIM Flag Definition

TIM_FLAG_UPDATE

TIM_FLAG_CC1

TIM_FLAG_CC2

TIM_FLAG_CC3

TIM_FLAG_CC4

TIM_FLAG_COM

TIM_FLAG_TRIGGER

TIM_FLAG_BREAK

TIM_FLAG_CC1OF

TIM_FLAG_CC2OF

TIM_FLAG_CC3OF

TIM_FLAG_CC4OF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

TIM_ICPOLARITY_FALLING

TIM 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 Lock level

TIM_LOCKLEVEL_OFF

TIM_LOCKLEVEL_1

TIM_LOCKLEVEL_2

TIM_LOCKLEVEL_3

TIM Master Mode Selection

TIM_TRGO_RESET

TIM_TRGO_ENABLE

TIM_TRGO_UPDATE

TIM_TRGO_OC1

TIM_TRGO_OC1REF

TIM_TRGO_OC2REF

TIM_TRGO_OC3REF

TIM_TRGO_OC4REF

TIM Master Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

TIM_MASTERSLAVEMODE_DISABLE

TIM One Pulse Mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

TIM OSSI Off State Selection for Idle mode state

TIM_OSSI_ENABLE

TIM_OSSI_DISABLE

TIM OSSR Off State Selection for Run mode state

TIM_OSSR_ENABLE

TIM_OSSR_DISABLE

TIM Output Compare and PWM modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

TIM_OCMODE_FORCED_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

TIM_OCNIDLESTATE_RESET

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

TIM_OCNPOLARITY_LOW

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

TIM_OUTPUTNSTATE_ENABLE

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

TIM_OUTPUTSTATE_ENABLE

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

TIM Slave Mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TIM TI1 Input 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

42 HAL TIM Extension Driver

42.1 TIMEx Firmware driver registers structures

42.1.1 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

42.1.2 TIM_BreakDeadTimeConfigTypeDef

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***
- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State This parameter can be a value of ***TIM_Break_Input_enable_disable***

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

42.1.3 TIM_MasterConfigTypeDef

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***

42.2 TIMEx Firmware driver API description

42.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

42.2.2 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_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      – Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      – Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:

- HAL_TIMEx_HallSensor_Init and HAL_TIMEx_ConfigCommutationEvent: to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).

5. Activate the TIM peripheral using one of the start functions:
   - Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   - Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   - Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   - Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

42.2.3 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()***

42.2.4 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()***

42.2.5 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()***

42.2.6 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()***

42.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.

This section contains the following APIs:

- ***HAL_TIMEx_ConfigCommutationEvent()***
- ***HAL_TIMEx_ConfigCommutationEvent_IT()***
- ***HAL_TIMEx_ConfigCommutationEvent_DMA()***
- ***HAL_TIMEx_ConfigBreakDeadTime()***
- ***HAL_TIMEx_MasterConfigSynchronization()***

42.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- ***HAL_TIMEx_CommutationCallback()***
- ***HAL_TIMEx_BreakCallback()***
- ***TIMEx_DMACommutationCplt()***

42.2.9 Extension Peripheral State 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_TIMEx_HallSensor_GetState()***

42.2.10 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)**
Function description	Initializes the TIM Hall Sensor Interface and create the associated handle.
Parameters	• **htim:** : TIM Encoder Interface handle • **sConfig:** : TIM Hall Sensor configuration structure
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_DeInit

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes the TIM Hall Sensor interface.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_MspInit

Function name	**void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)**

Function description	Initializes the TIM Hall Sensor MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIMEx_HallSensor_MspDeInit

Function name	**void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)**
Function description	DeInitializes TIM Hall Sensor MSP.
Parameters	• **htim:** : TIM handle
Return values	• **None:**

HAL_TIMEx_HallSensor_Start

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Hall Sensor Interface.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Hall sensor Interface.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)**
Function description	Starts the TIM Hall Sensor Interface in interrupt mode.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)**
Function description	Stops the TIM Hall Sensor Interface in interrupt mode.
Parameters	• **htim:** : TIM handle
Return values	• **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**

Function description Starts the TIM Hall Sensor Interface in DMA mode.

Parameters
- **htim:** : TIM Hall Sensor handle
- **pData:** : The destination Buffer address.
- **Length:** : The length of data to be transferred from TIM peripheral to memory.

Return values
- **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)**

Function description Stops the TIM Hall Sensor Interface in DMA mode.

Parameters
- **htim:** : TIM handle

Return values
- **HAL:** status

HAL_TIMEx_OCN_Start

Function name **HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the TIM Output Compare signal generation on the complementary output.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_OCN_Stop

Function name **HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Output Compare signal generation on the complementary output.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name
HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description
Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters
- **htim:** : TIM OC handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_OCN_Stop_IT

Function name
HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description
Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_OCN_Start_DMA

Function name
HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description
Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** : The source Buffer address.
- **Length:** : The length of data to be transferred from memory to TIM peripheral

Return values
- **HAL:** status

HAL_TIMEx_OCN_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters
- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Starts the PWM signal generation on the complementary output.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the PWM signal generation on the complementary output.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT**

(TIM_HandleTypeDef * htim, uint32_t Channel)

Function description: Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters:
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values:
- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name: **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description: Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters:
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values:
- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name: **HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**

Function description: Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters:
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** : The source Buffer address.
- **Length:** : The length of data to be transferred from memory to TIM peripheral

Return values:
- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name **HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function description Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters
- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values
- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name **HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Starts the TIM One Pulse signal generation on the complemetary output.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIMEx_OnePulseN_Stop

Function name **HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Stops the TIM One Pulse signal generation on the complementary output.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIMEx_OnePulseN_Start_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters
- **htim:** : TIM One Pulse handle

- **OutputChannel:** : TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIMEx_OnePulseN_Stop_IT

Function name **HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function description Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters
- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values
- **HAL:** status

HAL_TIMEx_ConfigCommutationEvent

Function name **HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)**

Function description Configure the TIM commutation event sequence.

Parameters
- **htim:** : TIM handle
- **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** : the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values
- **HAL:** status

Notes
- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutationEvent_IT

Function name
HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description
Configure the TIM commutation event sequence with interrupt.

Parameters
- **htim:** : TIM handle
- **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** : the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values
- **HAL:** status

Notes
- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutationEvent_DMA

Function name
HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description
Configure the TIM commutation event sequence with DMA.

Parameters
- **htim:** : TIM handle
- **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** : the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the

TRGI of the Interface Timer
– TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values
- **HAL:** status

Notes
- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- : The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_ConfigBreakDeadTime

Function name
HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description
Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

Parameters
- **htim:** : TIM handle
- **sBreakDeadTimeConfig:** : pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values
- **HAL:** status

HAL_TIMEx_MasterConfigSynchronization

Function name
HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)

Function description
Configures the TIM in master mode.

Parameters
- **htim:** : TIM handle.
- **sMasterConfig:** : pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values
- **HAL:** status

HAL_TIMEx_CommutationCallback

Function name
void HAL_TIMEx_CommutationCallback (TIM_HandleTypeDef * htim)

Function description
Hall commutation changed callback in non blocking mode.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIMEx_BreakCallback

Function name **void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)**

Function description Hall Break detection callback in non blocking mode.

Parameters
- **htim:** : TIM handle

Return values
- **None:**

HAL_TIMEx_HallSensor_GetState

Function name **HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * htim)**

Function description Return the TIM Hall Sensor interface state.

Parameters
- **htim:** : TIM Hall Sensor handle

Return values
- **HAL:** state

TIMEx_DMACommutationCplt

Function name **void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)**

Function description TIM DMA Commutation callback.

Parameters
- **hdma:** : pointer to DMA handle.

Return values
- **None:**

42.3 TIMEx Firmware driver defines

42.3.1 TIMEx

TIMEx Clock Filter

IS_TIM_DEADTIME BreakDead Time

43 HAL UART Generic Driver

43.1 UART Firmware driver registers structures

43.1.1 UART_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
  This member configures the UART communication baud rate. The baud rate is computed using the following formula:IntegerDivider = ((PCLKx) / (16 * (huart->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t UART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UART_Word_Length***
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of ***UART_Over_Sampling***. This feature is only available on STM32F100xx family, so OverSampling parameter should always be set to 16.

43.1.2 UART_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***

- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef gState***
- ***__IO HAL_UART_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef UART_HandleTypeDef::Init***
  UART communication parameters
- ***uint8_t* UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO 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
- ***__IO uint16_t UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t UART_HandleTypeDef::ErrorCode***
  UART Error code

43.2 UART Firmware driver API description

43.2.1 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 the UART pins (TX as alternate function pull-up, RX as alternate function Input).

c. NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
- Configure the USARTx interrupt priority.
- Enable the NVIC USART IRQ handle.

d. DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
- Declare a DMA handle structure for the Tx/Rx channel.
- Enable the DMAx interface clock.
- Configure the declared DMA handle structure with the required Tx/Rx parameters.
- Configure the DMA Tx/Rx channel.
- Associate the 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 channel.
- Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)

3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the huart Init structure.
4. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
5. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
6. For the LIN mode, initialize the UART registers by calling the HAL_LIN_Init() API.
7. For the Multi-Processor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.



The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive process.



These APIs (HAL_UART_Init() and HAL_HalfDuplex_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

Three operation modes are available within this driver:

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_UART_Transmit()
- Receive an amount of data in blocking mode using HAL_UART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_UART_Transmit_IT()
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_UART_Receive_IT()

- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_UART_Transmit_DMA()
- At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxHalfCpltCallback
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_UART_Receive_DMA()
- At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxHalfCpltCallback
- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback
- Pause the DMA Transfer using HAL_UART_DMAPause()
- Resume the DMA Transfer using HAL_UART_DMAResume()
- Stop the DMA Transfer using HAL_UART_DMAStop()

UART HAL driver macros list

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

- __HAL_UART_ENABLE: Enable the UART peripheral
- __HAL_UART_DISABLE: Disable the UART peripheral
- __HAL_UART_GET_FLAG : Check whether the specified UART flag is set or not
- __HAL_UART_CLEAR_FLAG : Clear the specified UART pending flag
- __HAL_UART_ENABLE_IT: Enable the specified UART interrupt
- __HAL_UART_DISABLE_IT: Disable the specified UART interrupt
- __HAL_UART_GET_IT_SOURCE: Check whether the specified UART interrupt has occurred or not



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



Additional remark: 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. The UART frame formats depend on the frame length defined by the M bit (8-bits or 9-bits). Refer to the product reference manual for details.

43.2.2 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 manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

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()***

43.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the UART asynchronous and Half duplex data transfers.

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - Non blocking mode: The communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UART_TxCpltCallback(), HAL_UART_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or receive process. The HAL_UART_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are:
   - HAL_UART_Transmit()
   - HAL_UART_Receive()
3. Non Blocking mode APIs with Interrupt are:
   - HAL_UART_Transmit_IT()
   - HAL_UART_Receive_IT()
   - HAL_UART_IRQHandler()
4. Non Blocking mode functions with DMA are:
   - HAL_UART_Transmit_DMA()
   - HAL_UART_Receive_DMA()
   - HAL_UART_DMAPause()
   - HAL_UART_DMAResume()

  - HAL_UART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in non blocking mode:
  - HAL_UART_TxHalfCpltCallback()
  - HAL_UART_TxCpltCallback()
  - HAL_UART_RxHalfCpltCallback()
  - HAL_UART_RxCpltCallback()
  - HAL_UART_ErrorCallback()



In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state HAL_UART_STATE_BUSY_TX_RX can't be useful.

This section contains the following APIs:

- ***HAL_UART_Transmit()***
- ***HAL_UART_Receive()***
- ***HAL_UART_Transmit_IT()***
- ***HAL_UART_Receive_IT()***
- ***HAL_UART_Transmit_DMA()***
- ***HAL_UART_Receive_DMA()***
- ***HAL_UART_DMAPause()***
- ***HAL_UART_DMAResume()***
- ***HAL_UART_DMAStop()***
- ***HAL_UART_Abort()***
- ***HAL_UART_AbortTransmit()***
- ***HAL_UART_AbortReceive()***
- ***HAL_UART_Abort_IT()***
- ***HAL_UART_AbortTransmit_IT()***
- ***HAL_UART_AbortReceive_IT()***
- ***HAL_UART_IRQHandler()***
- ***HAL_UART_TxCpltCallback()***
- ***HAL_UART_TxHalfCpltCallback()***
- ***HAL_UART_RxCpltCallback()***
- ***HAL_UART_RxHalfCpltCallback()***
- ***HAL_UART_ErrorCallback()***
- ***HAL_UART_AbortCpltCallback()***
- ***HAL_UART_AbortTransmitCpltCallback()***
- ***HAL_UART_AbortReceiveCpltCallback()***

43.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART:

- HAL_LIN_SendBreak() API can be helpful to transmit the break character.
- HAL_MultiProcessor_EnterMuteMode() API can be helpful to enter the UART in mute mode.
- HAL_MultiProcessor_ExitMuteMode() API can be helpful to exit the UART mute mode by software.
- HAL_HalfDuplex_EnableTransmitter() API to enable the UART transmitter and disables the UART receiver in Half Duplex mode
- HAL_HalfDuplex_EnableReceiver() API to enable the UART receiver and disables the UART transmitter in Half Duplex mode

This section contains the following APIs:

- *HAL_LIN_SendBreak()*
- *HAL_MultiProcessor_EnterMuteMode()*
- *HAL_MultiProcessor_ExitMuteMode()*
- *HAL_HalfDuplex_EnableTransmitter()*
- *HAL_HalfDuplex_EnableReceiver()*

43.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of UART communication process, return Peripheral Errors occurred during communication process

- HAL_UART_GetState() API can be helpful to check in run-time the state of the UART peripheral.
- HAL_UART_GetError() check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- *HAL_UART_GetState()*
- *HAL_UART_GetError()*

43.2.6 Detailed description of functions

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 create the associated handle.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

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 create the associated handle.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_LIN_Init

Function name **HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)**

Function description Initializes the LIN mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **BreakDetectLength:** Specifies the LIN break detection length. This parameter can be one of the following values:
  - UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection
  - UART_LINBREAKDETECTLENGTH_11B: 11-bit break detection

Return values
- **HAL:** status

HAL_MultiProcessor_Init

Function name **HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)**

Function description Initializes the Multi-Processor mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **Address:** USART address
- **WakeUpMethod:** specifies the USART wake-up method. This parameter can be one of the following values:
  - UART_WAKEUPMETHOD_IDLELINE: Wake-up by an idle line detection
  - UART_WAKEUPMETHOD_ADDRESSMARK: Wake-up by an address mark

Return values
- **HAL:** status

HAL_UART_DeInit

Function name **HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)**

Function description DeInitializes the UART peripheral.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_UART_MspInit

Function name **void HAL_UART_MspInit (UART_HandleTypeDef * huart)**

Function description UART MSP Init.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_MspDeInit

Function name **void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)**

Function description UART MSP DeInit.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_Transmit

Function name **HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Sends an amount of data in blocking mode.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_UART_Receive

Function name **HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function description Receive an amount of data in blocking mode.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_UART_Transmit_IT

Function name **HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**

Function description Sends an amount of data in non blocking mode.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer

- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_UART_Receive_IT

Function name **HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**

Function description Receives an amount of data in non blocking mode.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values
- **HAL:** status

HAL_UART_Transmit_DMA

Function name **HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**

Function description Sends an amount of data in non blocking mode.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

HAL_UART_Receive_DMA

Function name **HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**

Function description Receives an amount of data in non blocking mode.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values
- **HAL:** status

Notes
- When the UART parity is enabled (PCE = 1) the data received contain the parity bit.

HAL_UART_DMAPause

Function name **HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)**

Function description Pauses the DMA Transfer.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that

contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_UART_DMAResume

Function name **HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)**

Function description Resumes the DMA Transfer.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_UART_DMAStop

Function name **HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)**

Function description Stops the DMA Transfer.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_UART_Abort

Function name **HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)**

Function description Abort ongoing transfers (blocking mode).

Parameters
- **huart:** UART handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name **HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)**

Function description Abort ongoing Transmit transfer (blocking mode).

Parameters
- **huart:** UART handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name **HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)**

Function description Abort ongoing Receive transfer (blocking mode).

Parameters
- **huart:** UART handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name **HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)**

Function description Abort ongoing transfers (Interrupt mode).

Parameters
- **huart:** UART handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name **HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)**

Function description	Abort ongoing Transmit transfer (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name	**HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)**
Function description	Abort ongoing Receive transfer (Interrupt mode).
Parameters	• **huart:** UART handle.
Return values	• **HAL:** status
Notes	• This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP InterruptsDisable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback • This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name	**void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)**
Function description	This function handles UART interrupt request.
Parameters	• **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **None:**

HAL_UART_TxCpltCallback

Function name	**void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)**
Function description	Tx Transfer completed callbacks.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_TxHalfCpltCallback

Function name **void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)**

Function description Tx Half Transfer completed callbacks.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_RxCpltCallback

Function name **void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)**

Function description Rx Transfer completed callbacks.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_RxHalfCpltCallback

Function name **void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)**

Function description Rx Half Transfer completed callbacks.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_ErrorCallback

Function name **void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)**

Function description UART error callbacks.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **None:**

HAL_UART_AbortCpltCallback

Function name	**void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_AbortTransmitCpltCallback

Function name	**void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_UART_AbortReceiveCpltCallback

Function name	**void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)**
Function description	UART Abort Receive Complete callback.
Parameters	• **huart:** UART handle.
Return values	• **None:**

HAL_LIN_SendBreak

Function name	**HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)**
Function description	Transmits break characters.
Parameters	• **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name	**HAL_StatusTypeDef HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)**
Function description	Enters the UART in mute mode.
Parameters	• **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• **HAL:** status

HAL_MultiProcessor_ExitMuteMode

Function name **HAL_StatusTypeDef HAL_MultiProcessor_ExitMuteMode (UART_HandleTypeDef * huart)**

Function description Exits the UART mute mode: wake up software.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_HalfDuplex_EnableTransmitter

Function name **HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)**

Function description Enables the UART transmitter and disables the UART receiver.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name **HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)**

Function description Enables the UART receiver and disables the UART transmitter.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** status

HAL_UART_GetState

Function name **HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)**

Function description Returns the UART state.

Parameters
- **huart:** pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values
- **HAL:** state

HAL_UART_GetError

Function name **uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)**

Function description Return the UART error code.

Parameters
- **huart:** : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values
- **UART:** Error Code

43.3 UART Firmware driver defines

43.3.1 UART

UART Error Code

HAL_UART_ERROR_NONE	No error
HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	Frame error
HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA transfer error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:
- Reset UART handle gstate & RxState.

Parameters:
- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_FLUSH_DRREGISTER

Description:
- Flushs the UART DR register.

Parameters:
- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_GET_FLAG

Description:
- Checks whether the specified UART flag is set or not.

Parameters:
- __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4 or 5 to select the USART or UART peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_CTS: CTS Change flag (not available for

UART4 and UART5)
- UART_FLAG_LBD: LIN Break detection flag
- UART_FLAG_TXE: Transmit data register empty flag
- UART_FLAG_TC: Transmission Complete flag
- UART_FLAG_RXNE: Receive data register not empty flag
- UART_FLAG_IDLE: Idle Line detection flag
- UART_FLAG_ORE: OverRun Error flag
- UART_FLAG_NE: Noise Error flag
- UART_FLAG_FE: Framing Error flag
- UART_FLAG_PE: Parity Error flag

Return value:

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

__HAL_UART_CLEAR_FLAG

Description:

- Clears the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5).
  - UART_FLAG_LBD: LIN Break detection flag.
  - UART_FLAG_TC: Transmission Complete flag.
  - UART_FLAG_RXNE: Receive data register not empty flag.

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register followed by a

read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_UART_CLEAR_PEFLAG

Description:

- Clears the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_CLEAR_FEFLAG

Description:

- Clears the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_CLEAR_NEFLAG

Description:

- Clears the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_CLEAR_OREFLAG

Description:

- Clears the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clears the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_CTS: CTS change interrupt
  - UART_IT_LBD: LIN Break detection interrupt
  - UART_IT_TXE: Transmit Data Register empty interrupt
  - UART_IT_TC: Transmission complete interrupt
  - UART_IT_RXNE: Receive Data register not empty interrupt
  - UART_IT_IDLE: Idle line detection interrupt
  - UART_IT_PE: Parity Error interrupt
  - UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART_IT_CTS: CTS change interrupt
  - UART_IT_LBD: LIN Break

detection interrupt
- UART_IT_TXE: Transmit Data Register empty interrupt
- UART_IT_TC: Transmission complete interrupt
- UART_IT_RXNE: Receive Data register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_PE: Parity Error interrupt
- UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

__HAL_UART_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __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_ERR: Error interrupt

Return value:

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

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Notes:

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Notes:

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e __HAL_UART_ENABLE(__HANDLE

__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

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 any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

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_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

__HAL_UART_DISABLE

Description:

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

UART FLags

UART_FLAG_CTS

UART_FLAG_LBD

UART_FLAG_TXE

UART_FLAG_TC

UART_FLAG_RXNE

UART_FLAG_IDLE

UART_FLAG_ORE

UART_FLAG_NE

UART_FLAG_FE

UART_FLAG_PE

UART Hardware Flow Control

UART_HWCONTROL_NONE

UART_HWCONTROL_RTS

UART_HWCONTROL_CTS

UART_HWCONTROL_RTS_CTS

UART Interrupt Definitions

UART_IT_PE

UART_IT_TXE

UART_IT_TC

UART_IT_RXNE

UART_IT_IDLE

UART_IT_LBD

UART_IT_CTS

UART_IT_ERR

UART LIN Break Detection Length

UART_LINBREAKDETECTLENGTH_10B

UART_LINBREAKDETECTLENGTH_11B

UART Transfer Mode

UART_MODE_RX

UART_MODE_TX

UART_MODE_TX_RX

UART Over Sampling

UART_OVERSAMPLING_16

UART Parity

UART_PARITY_NONE

UART_PARITY_EVEN

UART_PARITY_ODD

UART State

UART_STATE_DISABLE

UART_STATE_ENABLE

UART Number of Stop Bits

UART_STOPBITS_1

UART_STOPBITS_2

UART Wakeup Functions

UART_WAKEUPMETHOD_IDLELINE

UART_WAKEUPMETHOD_ADDRESSMARK

UART Word Length

UART_WORDLENGTH_8B

UART_WORDLENGTH_9B

44 HAL USART Generic Driver

44.1 USART Firmware driver registers structures

44.1.1 USART_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***
  This member configures the Usart communication baud rate. The baud rate is computed using the following formula:IntegerDivider = ((PCLKx) / (16 * (husart->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t USART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_Word_Length***
- ***uint32_t USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***
- ***uint32_t USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***
- ***uint32_t USART_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***
- ***uint32_t USART_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***
- ***uint32_t USART_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***

44.1.2 USART_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***

- ***USART_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_USART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* USART_HandleTypeDef::Instance***
  USART 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
- ***__IO 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
- ***__IO uint16_t USART_HandleTypeDef::RxXferCount***
  Usart Rx Transfer Counter
- ***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
- ***__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State***
  Usart communication state
- ***__IO uint32_t USART_HandleTypeDef::ErrorCode***
  USART Error code

44.2 USART Firmware driver API description

44.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure.
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit () API:
   a. Enable the USARTx interface clock.
   b. USART pins configuration:
      – Enable the clock for the USART GPIOs.

  - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).

c. NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
  - Configure the USARTx interrupt priority.
  - Enable the NVIC USART IRQ handle.

d. DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
  - Declare a DMA handle structure for the Tx/Rx channel.
  - Enable the DMAx interface clock.
  - Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx channel.
  - Associate the 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 channel.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
  - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)

3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the husart Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
  - These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API. The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_USART_Transmit()
- Receive an amount of data in blocking mode using HAL_USART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_USART_Transmit_IT()
- At transmission end of transfer HAL_USART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_USART_Receive_IT()
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_USART_Transmit_DMA()
- At transmission end of half transfer HAL_USART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxHalfCpltCallback
- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_USART_Receive_DMA()
- At reception end of half transfer HAL_USART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxHalfCpltCallback
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback
- Pause the DMA Transfer using HAL_USART_DMAPause()
- Resume the DMA Transfer using HAL_USART_DMAResume()
- Stop the DMA Transfer using HAL_USART_DMAStop()

USART HAL driver macros list

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

- __HAL_USART_ENABLE: Enable the USART peripheral
- __HAL_USART_DISABLE: Disable the USART peripheral
- __HAL_USART_GET_FLAG : Check whether the specified USART flag is set or not
- __HAL_USART_CLEAR_FLAG : Clear the specified USART pending flag
- __HAL_USART_ENABLE_IT: Enable the specified USART interrupt
- __HAL_USART_DISABLE_IT: Disable the specified USART interrupt



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



Additional remark: 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. The USART frame formats depend on the frame length defined by the M bit (8-bits or 9-bits). Refer to the product reference manual for details.”

44.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length

- Stop Bit
- Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible USART frame formats.
- USART polarity
- USART phase
- USART LastBit
- Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

This section contains the following APIs:

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***

44.2.3 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
   - HAL_USART_Transmit() in simplex mode
   - HAL_USART_Receive() in full duplex receive only
   - HAL_USART_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
   - HAL_USART_Transmit_IT()in simplex mode
   - HAL_USART_Receive_IT() in full duplex receive only
   - HAL_USART_TransmitReceive_IT() in full duplex mode
   - HAL_USART_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_USART_Transmit_DMA()in simplex mode
   - HAL_USART_Receive_DMA() in full duplex receive only
   - HAL_USART_TransmitReceive_DMA() in full duplex mode
   - HAL_USART_DMAPause()
   - HAL_USART_DMAResume()
   - HAL_USART_DMAStop()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_USART_TxHalfCpltCallback()
   - HAL_USART_TxCpltCallback()
   - HAL_USART_RxHalfCpltCallback()
   - HAL_USART_RxCpltCallback()
   - HAL_USART_ErrorCallback()
   - HAL_USART_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_Abort()***
- ***HAL_USART_Abort_IT()***
- ***HAL_USART_IRQHandler()***
- ***HAL_USART_TxCpltCallback()***
- ***HAL_USART_TxHalfCpltCallback()***
- ***HAL_USART_RxCpltCallback()***
- ***HAL_USART_RxHalfCpltCallback()***
- ***HAL_USART_TxRxCpltCallback()***
- ***HAL_USART_ErrorCallback()***
- ***HAL_USART_AbortCpltCallback()***

44.2.4 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()***

44.2.5 Detailed description of functions

HAL_USART_Init

Function name	**HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)**
Function description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **HAL:** status

HAL_USART_DeInit

Function name **HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)**

Function description DeInitializes the USART peripheral.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **HAL:** status

HAL_USART_MspInit

Function name **void HAL_USART_MspInit (USART_HandleTypeDef * husart)**

Function description USART MSP Init.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_MspDeInit

Function name **void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)**

Function description USART MSP DeInit.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_Transmit

Function name **HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)**

Function description Simplex Send an amount of data in blocking mode.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_USART_Receive

Function name **HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**

Function description Full-Duplex Receive an amount of data in blocking mode.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pRxData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values
- **HAL:** status

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 receive an amount of data in full-duplex mode (blocking mode).

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData:** Pointer to data transmitted buffer
- **pRxData:** Pointer to data received buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- **HAL:** status

HAL_USART_Transmit_IT

Function name **HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**

Function description Simplex Send an amount of data in non-blocking mode.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values
- **HAL:** status

Notes
- The USART errors are not managed to avoid the overrun error.

HAL_USART_Receive_IT

Function name **HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t**

Size)

Function description: Simplex Receive an amount of data in non-blocking mode.

Parameters:
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pRxData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values:
- **HAL:** status

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 receive an amount of data in full-duplex mode (non-blocking).

Parameters:
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData:** Pointer to data transmitted buffer
- **pRxData:** Pointer to data received buffer
- **Size:** Amount of data to be received

Return values:
- **HAL:** status

HAL_USART_Transmit_DMA

Function name: **HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**

Function description: Simplex Send an amount of data in non-blocking mode.

Parameters:
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values:
- **HAL:** status

HAL_USART_Receive_DMA

Function name: **HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**

Function description: Full-Duplex Receive an amount of data in non-blocking mode.

Parameters:
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pRxData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values
- **HAL:** status

Notes
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

HAL_USART_TransmitReceive_DMA

Function name **HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**

Function description Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData:** Pointer to data transmitted buffer
- **pRxData:** Pointer to data received buffer
- **Size:** Amount of data to be received

Return values
- **HAL:** status

Notes
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

HAL_USART_DMAPause

Function name **HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)**

Function description Pauses the DMA Transfer.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **HAL:** status

HAL_USART_DMAResume

Function name **HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)**

Function description Resumes the DMA Transfer.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **HAL:** status

HAL_USART_DMAStop

Function name **HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)**

Function description Stops the DMA Transfer.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **HAL:** status

HAL_USART_Abort

Function name **HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)**

Function description Abort ongoing transfer (blocking mode).

Parameters
- **husart:** USART handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer (either Tx or Rx, as described by TransferType parameter) started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name **HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)**

Function description Abort ongoing transfer (Interrupt mode).

Parameters
- **husart:** USART handle.

Return values
- **HAL:** status

Notes
- This procedure could be used for aborting any ongoing transfer (either Tx or Rx, as described by TransferType parameter) started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name **void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)**

Function description This function handles USART interrupt request.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_TxCpltCallback

Function name **void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)**

Function description Tx Transfer completed callbacks.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_TxHalfCpltCallback

Function name **void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)**

Function description Tx Half Transfer completed callbacks.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_RxCpltCallback

Function name **void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)**

Function description Rx Transfer completed callbacks.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_RxHalfCpltCallback

Function name **void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)**

Function description Rx Half Transfer completed callbacks.

Parameters
- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values
- **None:**

HAL_USART_TxRxCpltCallback

Function name	**void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)**
Function description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	• **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• **None:**

HAL_USART_ErrorCallback

Function name	**void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)**
Function description	USART error callbacks.
Parameters	• **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• **None:**

HAL_USART_AbortCpltCallback

Function name	**void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)**
Function description	USART Abort Complete callback.
Parameters	• **husart:** USART handle.
Return values	• **None:**

HAL_USART_GetState

Function name	**HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)**
Function description	Returns the USART state.
Parameters	• **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• **HAL:** state

HAL_USART_GetError

Function name	**uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)**
Function description	Return the USART error code.
Parameters	• **husart:** : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values
- **USART:** Error Code

44.3 USART Firmware driver defines

44.3.1 USART

USART Clock

USART_CLOCK_DISABLE

USART_CLOCK_ENABLE

USART Clock Phase

USART_PHASE_1EDGE

USART_PHASE_2EDGE

USART Clock Polarity

USART_POLARITY_LOW

USART_POLARITY_HIGH

USART Error Code

HAL_USART_ERROR_NONE	No error
HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	Frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_GET_FLAG

Description:

- Checks whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – USART_FLAG_TXE: Transmit data

register empty flag
  - USART_FLAG_TC: Transmission Complete flag
  - USART_FLAG_RXNE: Receive data register not empty flag
  - USART_FLAG_IDLE: Idle Line detection flag
  - USART_FLAG_ORE: OverRun Error flag
  - USART_FLAG_NE: Noise Error flag
  - USART_FLAG_FE: Framing Error flag
  - USART_FLAG_PE: Parity Error flag

Return value:

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

__HAL_USART_CLEAR_FLAG

Description:

- Clears the specified USART pending flags.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_FLAG_TC: Transmission Complete flag.
  - USART_FLAG_RXNE: Receive data register not empty flag.

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register TXE flag is cleared only by a write to the USART_DR register

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupts.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __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) This parameter can be: ENABLE or DISABLE.

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupts.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __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) This parameter can be: ENABLE or DISABLE.

__HAL_USART_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_TXE: Transmit Data Register empty interrupt
  - USART_IT_TC: Transmission complete interrupt
  - USART_IT_RXNE: Receive Data register not empty interrupt
  - USART_IT_IDLE: Idle line detection interrupt
  - USART_IT_ERR: Error interrupt
  - USART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

USART Flags

USART_FLAG_TXE

USART_FLAG_TC

USART_FLAG_RXNE

USART_FLAG_IDLE

USART_FLAG_ORE

USART_FLAG_NE

USART_FLAG_FE

USART_FLAG_PE

USART Interrupts Definition

USART_IT_PE

USART_IT_TXE

USART_IT_TC

USART_IT_RXNE

USART_IT_IDLE

USART_IT_LBD

USART_IT_CTS

USART_IT_ERR

USART Last Bit

USART_LASTBIT_DISABLE

USART_LASTBIT_ENABLE

USART Mode

USART_MODE_RX

USART_MODE_TX

USART_MODE_TX_RX

USART NACK State

USART_NACK_ENABLE

USART_NACK_DISABLE

USART Parity

USART_PARITY_NONE

USART_PARITY_EVEN

USART_PARITY_ODD

USART Number of Stop Bits

USART_STOPBITS_1

USART_STOPBITS_0_5

USART_STOPBITS_2

USART_STOPBITS_1_5

USART Word Length

USART_WORDLENGTH_8B

USART_WORDLENGTH_9B

45 HAL WWDG Generic Driver

45.1 WWDG Firmware driver registers structures

45.1.1 WWDG_InitTypeDef

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

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 Min_Data = 0x40 and Max_Data = 0x7F
- ***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
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

45.1.2 WWDG_HandleTypeDef

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDef::Init***
  WWDG required parameters

45.2 WWDG Firmware driver API description

45.2.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register can be used to inform when a WWDG reset occurs.

- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- 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 36 MHz(PCLK1): 910 us / 58.25 ms
- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches the value 0x40, an 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. In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case, the corresponding interrupt service routine (ISR) should reload the WWDG counter to avoid the WWDG reset, then trigger the required actions. Note:When the EWI interrupt cannot be served, e.g. due to a system lock in a higher priority task, the WWDG reset will eventually be generated.
- Debug mode : When the microcontroller enters debug mode (core halted), the WWDG counter either continues to work normally or stops, depending on DBG_WWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_WWDG() and __HAL_DBGMCU_UNFREEZE_WWDG() macros

45.2.2 How to use this driver

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Set the WWDG prescaler, refresh window, counter value and Early Wakeup Interrupt mode using using HAL_WWDG_Init() function. This enables WWDG peripheral and the downcounter starts downcounting from given counter value. Init function can be called again to modify all watchdog parameters, however if EWI mode has been set once, it can't be clear until next reset.
- 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 window value already programmed.
- if Early Wakeup Interrupt mode is enable an interrupt is generated when the counter reaches 0x40. User can add his own code in weak function HAL_WWDG_EarlyWakeupCallback().

WWDG HAL driver macros list

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

- __HAL_WWDG_GET_IT_SOURCE: Check the selected WWDG's interrupt source.
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status.
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags.

45.2.3 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the WWDG_InitTypeDef of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- ***HAL_WWDG_Init()***
- ***HAL_WWDG_MspInit()***

45.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- ***HAL_WWDG_Refresh()***
- ***HAL_WWDG_IRQHandler()***
- ***HAL_WWDG_EarlyWakeupCallback()***

45.2.5 Detailed description of functions

HAL_WWDG_Init

Function name **HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)**

Function description Initialize the WWDG according to the specified.

Parameters
- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values
- **HAL:** status

HAL_WWDG_MspInit

Function name **void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)**

Function description Initialize the WWDG MSP.

Parameters
- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values
- **None:**

Notes
- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_Refresh

Function name **HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)**

Function description Refresh the WWDG.

Parameters
- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values
- **HAL:** status

HAL_WWDG_IRQHandler

Function name **void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)**

Function description Handle WWDG interrupt request.

Parameters
- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values
- **None:**

Notes
- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. 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.

HAL_WWDG_EarlyWakeupCallback

Function name **void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)**

Function description WWDG Early Wakeup callback.

Parameters
- **hwwdg:** : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values
- **None:**

45.3 WWDG Firmware driver defines

45.3.1 WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE EWI Disable

WWDG_EWI_ENABLE EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:
- Enables the WWDG peripheral.

Parameters:
- __HANDLE__: WWDG handle

Return value:
- None

__HAL_WWDG_ENABLE_IT

Description:
- Enables the WWDG early wakeup interrupt.

Parameters:

- __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_GET_IT

Description:

- Checks whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: 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 bits to clear the selected interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

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

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

46 LL ADC Generic Driver

46.1 ADC Firmware driver registers structures

46.1.1 LL_ADC_CommonInitTypeDef

Data Fields

- ***uint32_t Multimode***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::Multimode***
  Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances). This parameter can be a value of ***ADC_LL_EC_MULTI_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetMultimode()**.

46.1.2 LL_ADC_InitTypeDef

Data Fields

- ***uint32_t DataAlignment***
- ***uint32_t SequencersScanMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::DataAlignment***
  Set ADC conversion data alignment. This parameter can be a value of ***ADC_LL_EC_DATA_ALIGN***This feature can be modified afterwards using unitary function **LL_ADC_SetDataAlignment()**.
- ***uint32_t LL_ADC_InitTypeDef::SequencersScanMode***
  Set ADC scan selection. This parameter can be a value of ***ADC_LL_EC_SCAN_SELECTION***This feature can be modified afterwards using unitary function **LL_ADC_SetSequencersScanMode()**.

46.1.3 LL_ADC_REG_InitTypeDef

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DMATransfer***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***
  Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***
  Note: On this STM32 serie, external trigger is set with trigger polarity: rising edge (only trigger polarity available on this STM32 serie). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***
  Set ADC group regular sequencer length. This parameter can be a value of

ADC_LL_EC_REG_SEQ_SCAN_LENGTH
Note: This parameter is discarded if scan mode is disabled (refer to parameter 'ADC_SequencersScanMode'). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***
  Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***
  Note: This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***
  Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode.This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::DMATransfer***
  Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode. This parameter can be a value of ***ADC_LL_EC_REG_DMA_TRANSFER***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDMATransfer()**.

46.1.4 LL_ADC_INJ_InitTypeDef

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t TrigAuto***

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***
  Set ADC group injected conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_INJ_TRIGGER_SOURCE***
  Note: On this STM32 serie, external trigger is set with trigger polarity: rising edge (only trigger polarity available on this STM32 serie). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***
  Set ADC group injected sequencer length. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_SCAN_LENGTH***
  Note: This parameter is discarded if scan mode is disabled (refer to parameter 'ADC_SequencersScanMode'). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***
  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_DISCONT_MODE***
  Note: This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***
  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of ***ADC_LL_EC_INJ_TRIG_AUTO*** Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger.This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

46.2 ADC Firmware driver API description

46.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (ADC_TypeDef * ADCx, uint32_t Register)**
Function description	Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.
Parameters	• **ADCx:** ADC instance • **Register:** This parameter can be one of the following values: (1) Available on devices with several ADC instances. – LL_ADC_DMA_REG_REGULAR_DATA – LL_ADC_DMA_REG_REGULAR_DATA_MULTI (1)
Return values	• **ADC:** register address
Notes	• These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request. • This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY); • For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter. • On STM32F1, only ADC instances ADC1 and ADC3 have DMA transfer capability, not ADC2 (ADC2 and ADC3 instances not available on all devices). • On STM32F1, multimode can be used only with ADC1 and ADC2, not ADC3. Therefore, the corresponding parameter of data transfer for multimode can be used only with ADC1 and ADC2. (ADC2 and ADC3 instances not available on all devices).
Reference Manual to LL API cross reference:	• DR DATA LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonPathInternalCh

Function name	**__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)**
Function description	Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **PathInternal:** This parameter can be a combination of the following values: – LL_ADC_PATH_INTERNAL_NONE – LL_ADC_PATH_INTERNAL_VREFINT – LL_ADC_PATH_INTERNAL_TEMPSENSOR
Return values	• **None:**
Notes	• One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR) • Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US. • ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.
Reference Manual to LL API cross reference:	• CR2 TSVREFE LL_ADC_SetCommonPathInternalCh

LL_ADC_GetCommonPathInternalCh

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **Returned:** value can be a combination of the following values: – LL_ADC_PATH_INTERNAL_NONE – LL_ADC_PATH_INTERNAL_VREFINT – LL_ADC_PATH_INTERNAL_TEMPSENSOR
Notes	• One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:	• CR2 TSVREFE LL_ADC_GetCommonPathInternalCh

LL_ADC_SetDataAlignment

Function name	**__STATIC_INLINE void LL_ADC_SetDataAlignment (ADC_TypeDef * ADCx, uint32_t DataAlignment)**
Function description	Set ADC conversion data alignment.
Parameters	• **ADCx:** ADC instance • **DataAlignment:** This parameter can be one of the following values: – LL_ADC_DATA_ALIGN_RIGHT – LL_ADC_DATA_ALIGN_LEFT
Return values	• **None:**
Notes	• Refer to reference manual for alignments formats dependencies to ADC resolutions.
Reference Manual to LL API cross reference:	• CR2 ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name	**__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (ADC_TypeDef * ADCx)**
Function description	Get ADC conversion data alignment.
Parameters	• **ADCx:** ADC instance
Return values	• **Returned:** value can be one of the following values: – LL_ADC_DATA_ALIGN_RIGHT – LL_ADC_DATA_ALIGN_LEFT
Notes	• Refer to reference manual for alignments formats dependencies to ADC resolutions.
Reference Manual to LL API cross reference:	• CR2 ALIGN LL_ADC_SetDataAlignment

LL_ADC_SetSequencersScanMode

Function name	**__STATIC_INLINE void LL_ADC_SetSequencersScanMode (ADC_TypeDef * ADCx, uint32_t ScanMode)**
Function description	Set ADC sequencers scan mode, for all ADC groups (group regular, group injected).
Parameters	• **ADCx:** ADC instance • **ScanMode:** This parameter can be one of the following values: – LL_ADC_SEQ_SCAN_DISABLE – LL_ADC_SEQ_SCAN_ENABLE

Return values
- **None:**

Notes
- According to sequencers scan mode : If disabled: ADC conversion is performed in unitary conversion mode (one channel converted, that defined in rank 1). Configuration of sequencers of all ADC groups (sequencer scan length, ...) is discarded: equivalent to scan length of 1 rank.If enabled: ADC conversions are performed in sequence conversions mode, according to configuration of sequencers of each ADC group (sequencer scan length, ...). Refer to function LL_ADC_REG_SetSequencerLength() and to function LL_ADC_INJ_SetSequencerLength().

Reference Manual to LL API cross reference:
- CR1 SCAN LL_ADC_SetSequencersScanMode

LL_ADC_GetSequencersScanMode

Function name
__STATIC_INLINE uint32_t LL_ADC_GetSequencersScanMode (ADC_TypeDef * ADCx)

Function description
Get ADC sequencers scan mode, for all ADC groups (group regular, group injected).

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  - LL_ADC_SEQ_SCAN_DISABLE
  - LL_ADC_SEQ_SCAN_ENABLE

Notes
- According to sequencers scan mode : If disabled: ADC conversion is performed in unitary conversion mode (one channel converted, that defined in rank 1). Configuration of sequencers of all ADC groups (sequencer scan length, ...) is discarded: equivalent to scan length of 1 rank.If enabled: ADC conversions are performed in sequence conversions mode, according to configuration of sequencers of each ADC group (sequencer scan length, ...). Refer to function LL_ADC_REG_SetSequencerLength() and to function LL_ADC_INJ_SetSequencerLength().

Reference Manual to LL API cross reference:
- CR1 SCAN LL_ADC_GetSequencersScanMode

LL_ADC_REG_SetTriggerSource

Function name
__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description
Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).

Parameters
- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values: (1) On STM32F1, parameter available on all ADC instances: ADC1, ADC2, ADC3 (for ADC instances ADCx available on the selected device).

  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3 (1)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2 (2)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2 (2)
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO (2)
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4 (2)
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11 (2)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO (2)(4)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO_ADC3 (3)
  - LL_ADC_REG_TRIG_EXT_TIM3_CH1 (3)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH3 (3)
  - LL_ADC_REG_TRIG_EXT_TIM8_CH1 (3)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO (3)
  - LL_ADC_REG_TRIG_EXT_TIM5_CH1 (3)
  - LL_ADC_REG_TRIG_EXT_TIM5_CH3 (3)
- (2) On STM32F1, parameter available only on ADC instances: ADC1, ADC2 (for ADC instances ADCx available on the selected device).
- (3) On STM32F1, parameter available only on ADC instances: ADC3 (for ADC instances ADCx available on the selected device).
- (4) On STM32F1, parameter available only on high-density and XL-density devices. A remap of trigger must be done at top level (refer to AFIO peripheral).

Return values
- **None:**

Notes
- On this STM32 serie, external trigger is set with trigger polarity: rising edge (only trigger polarity available on this STM32 serie).
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR2 EXTSEL LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name
__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (ADC_TypeDef * ADCx)

Function description
Get ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values: (1) On STM32F1, parameter available on all ADC instances: ADC1, ADC2, ADC3 (for ADC instances ADCx available on the selected device).
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3 (1)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2 (2)

  - LL_ADC_REG_TRIG_EXT_TIM2_CH2 (2)
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO (2)
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4 (2)
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11 (2)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO (2)(4)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO_ADC3 (3)
  - LL_ADC_REG_TRIG_EXT_TIM3_CH1 (3)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH3 (3)
  - LL_ADC_REG_TRIG_EXT_TIM8_CH1 (3)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO (3)
  - LL_ADC_REG_TRIG_EXT_TIM5_CH1 (3)
  - LL_ADC_REG_TRIG_EXT_TIM5_CH3 (3)
- (2) On STM32F1, parameter available only on ADC instances: ADC1, ADC2 (for ADC instances ADCx available on the selected device).
- (3) On STM32F1, parameter available only on ADC instances: ADC3 (for ADC instances ADCx available on the selected device).
- (4) On STM32F1, parameter available only on high-density and XL-density devices. A remap of trigger must be done at top level (refer to AFIO peripheral).

Notes
- To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR2 EXTSEL LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name
__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)

Function description
Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters
- **ADCx:** ADC instance

Return values
- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes
- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().

Reference Manual to LL API cross reference:
- CR2 EXTSEL LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Return values

- **None:**

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- On this STM32 serie, group regular sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function

LL_ADC_SetSequencersScanMode().
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:
- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name
__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (ADC_TypeDef * ADCx)

Function description
Get ADC group regular sequencer length and scan direction.

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Notes
- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".

- On this STM32 serie, group regular sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL_ADC_SetSequencersScanMode().
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Return values

- **None:**

Notes

- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CR1 DISCEN LL_ADC_REG_SetSequencerDiscont
- CR1 DISCNUM LL_ADC_REG_SetSequencerDiscont

LL_ADC_REG_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:

- LL_ADC_REG_SEQ_DISCONT_DISABLE
- LL_ADC_REG_SEQ_DISCONT_1RANK
- LL_ADC_REG_SEQ_DISCONT_2RANKS
- LL_ADC_REG_SEQ_DISCONT_3RANKS
- LL_ADC_REG_SEQ_DISCONT_4RANKS
- LL_ADC_REG_SEQ_DISCONT_5RANKS
- LL_ADC_REG_SEQ_DISCONT_6RANKS
- LL_ADC_REG_SEQ_DISCONT_7RANKS
- LL_ADC_REG_SEQ_DISCONT_8RANKS

Reference Manual to LL API cross reference:

- CR1 DISCEN LL_ADC_REG_GetSequencerDiscont
- CR1 DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name **__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)**

Function description Set ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16
- **Channel:** This parameter can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10

- LL_ADC_CHANNEL_11
- LL_ADC_CHANNEL_12
- LL_ADC_CHANNEL_13
- LL_ADC_CHANNEL_14
- LL_ADC_CHANNEL_15
- LL_ADC_CHANNEL_16
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_VREFINT (1)
- LL_ADC_CHANNEL_TEMPSENSOR (1)

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 serie, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().

Reference Manual to LL API cross reference:

- SQR3 SQ1 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ2 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ3 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ4 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ5 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ6 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ10 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ11 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ12 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ13 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ14 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ15 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1

- LL_ADC_REG_RANK_2
- LL_ADC_REG_RANK_3
- LL_ADC_REG_RANK_4
- LL_ADC_REG_RANK_5
- LL_ADC_REG_RANK_6
- LL_ADC_REG_RANK_7
- LL_ADC_REG_RANK_8
- LL_ADC_REG_RANK_9
- LL_ADC_REG_RANK_10
- LL_ADC_REG_RANK_11
- LL_ADC_REG_RANK_12
- LL_ADC_REG_RANK_13
- LL_ADC_REG_RANK_14
- LL_ADC_REG_RANK_15
- LL_ADC_REG_RANK_16

Return values

- **Returned:** value can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
- (1) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- On this STM32 serie, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with

parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- SQR3 SQ1 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ2 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ3 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ4 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ5 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ6 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ10 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ11 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ12 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ13 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ14 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ15 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_REG_SetContinuousMode

Function name **__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)**

Function description Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CR2 CONT LL_ADC_REG_SetContinuousMode

LL_ADC_REG_GetContinuousMode

Function name **__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (ADC_TypeDef * ADCx)**

Function description Get ADC continuous conversion mode on ADC group regular.

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  – LL_ADC_REG_CONV_SINGLE
  – LL_ADC_REG_CONV_CONTINUOUS

Notes
- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.

Reference Manual to LL API cross reference:
- CR2 CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDMATransfer

Function name **__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)**

Function description Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters
- **ADCx:** ADC instance
- **DMATransfer:** This parameter can be one of the following values:
  – LL_ADC_REG_DMA_TRANSFER_NONE
  – LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Return values
- **None:**

Notes
- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:
- CR2 DMA LL_ADC_REG_SetDMATransfer

LL_ADC_REG_GetDMATransfer

Function name **__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (ADC_TypeDef * ADCx)**

Function description Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  - LL_ADC_REG_DMA_TRANSFER_NONE
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Notes
- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:
- CR2 DMA LL_ADC_REG_GetDMATransfer

LL_ADC_INJ_SetTriggerSource

Function name
__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description
Set ADC group injected conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).

Parameters
- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values: (1) On STM32F1, parameter available on all ADC instances: ADC1, ADC2, ADC3 (for ADC instances ADCx available on the selected device).
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (2)(4)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4_ADC3 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH3 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH2 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM5_TRGO (3)
  - LL_ADC_INJ_TRIG_EXT_TIM5_CH4 (3)
- (2) On STM32F1, parameter available only on ADC instances: ADC1, ADC2 (for ADC instances ADCx available

on the selected device).
- (3) On STM32F1, parameter available only on ADC instances: ADC3 (for ADC instances ADCx available on the selected device).
- (4) On STM32F1, parameter available only on high-density and XL-density devices. A remap of trigger must be done at top level (refer to AFIO peripheral).

Return values
- **None:**

Notes
- On this STM32 serie, external trigger is set with trigger polarity: rising edge (only trigger polarity available on this STM32 serie).
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR2 JEXTSEL LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name
__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (ADC_TypeDef * ADCx)

Function description
Get ADC group injected conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values: (1) On STM32F1, parameter available on all ADC instances: ADC1, ADC2, ADC3 (for ADC instances ADCx available on the selected device).
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (2)(4)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4_ADC3 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH3 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH2 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (3)
  - LL_ADC_INJ_TRIG_EXT_TIM5_TRGO (3)
  - LL_ADC_INJ_TRIG_EXT_TIM5_CH4 (3)
- (2) On STM32F1, parameter available only on ADC instances: ADC1, ADC2 (for ADC instances ADCx available on the selected device).
- (3) On STM32F1, parameter available only on ADC instances: ADC3 (for ADC instances ADCx available on the selected device).
- (4) On STM32F1, parameter available only on high-density and XL-density devices. A remap of trigger must be done at

top level (refer to AFIO peripheral).

Notes
- To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR2 JEXTSEL LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name **__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)**

Function description Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters
- **ADCx:** ADC instance

Return values
- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes
- In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.

Reference Manual to LL API cross reference:
- CR2 JEXTSEL LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetSequencerLength

Function name **__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)**

Function description Set ADC group injected sequencer length and scan direction.

Parameters
- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Return values
- **None:**

Notes
- This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- On this STM32 serie, group injected sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all

groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL_ADC_SetSequencersScanMode().
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:
- JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name
__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (ADC_TypeDef * ADCx)

Function description
Get ADC group injected sequencer length and scan direction.

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Notes
- This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- On this STM32 serie, group injected sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL_ADC_SetSequencersScanMode().
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:
- JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name
__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description
Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters
- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Return values
- **None:**

Notes
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:
- CR1 DISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name
__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (ADC_TypeDef * ADCx)

Function description
Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters
- **ADCx:** ADC instance

Return values
- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Reference Manual to LL API cross reference:
- CR1 DISCEN LL_ADC_REG_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name
__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description
Set ADC group injected sequence: channel on the selected sequence rank.

Parameters
- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **Channel:** This parameter can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13

- LL_ADC_CHANNEL_14
- LL_ADC_CHANNEL_15
- LL_ADC_CHANNEL_16
- LL_ADC_CHANNEL_17
- LL_ADC_CHANNEL_VREFINT (1)
- LL_ADC_CHANNEL_TEMPSENSOR (1)

Return values
- **None:**

Notes
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().

Reference Manual to LL API cross reference:
- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_GetSequencerRanks

Function name
__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)

Function description
Get ADC group injected sequence: channel on the selected sequence rank.

Parameters
- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values
- **Returned:** value can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16

  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
- (1) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:
- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name
__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description
Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters
- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values
- **None:**

Notes
- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.
- If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:
- CR1 JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name: **__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (ADC_TypeDef * ADCx)**

Function description: Get ADC group injected conversion trigger: independent or from ADC group regular.

Parameters:
- **ADCx:** ADC instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Reference Manual to LL API cross reference:
- CR1 JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_INJ_SetOffset

Function name: **__STATIC_INLINE void LL_ADC_INJ_SetOffset (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t OffsetLevel)**

Function description: Set ADC group injected offset.

Parameters:
- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values:
- **None:**

Notes:
- It sets: ADC group injected rank to which the offset programmed will be appliedOffset level (offset to be subtracted from the raw converted data). Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.
- Offset cannot be enabled or disabled. To emulate offset disabled, set an offset value equal to 0.

Reference Manual to LL API cross reference:
- JOFR1 JOFFSET1 LL_ADC_INJ_SetOffset
- JOFR2 JOFFSET2 LL_ADC_INJ_SetOffset
- JOFR3 JOFFSET3 LL_ADC_INJ_SetOffset
- JOFR4 JOFFSET4 LL_ADC_INJ_SetOffset

LL_ADC_INJ_GetOffset

Function name: **__STATIC_INLINE uint32_t LL_ADC_INJ_GetOffset (ADC_TypeDef * ADCx, uint32_t Rank)**

Function description: Get ADC group injected offset.

Parameters:
- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values:
- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes:
- It gives offset level (offset to be subtracted from the raw converted data). Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.

Reference Manual to LL API cross reference:
- JOFR1 JOFFSET1 LL_ADC_INJ_GetOffset
- JOFR2 JOFFSET2 LL_ADC_INJ_GetOffset
- JOFR3 JOFFSET3 LL_ADC_INJ_GetOffset
- JOFR4 JOFFSET4 LL_ADC_INJ_GetOffset

LL_ADC_SetChannelSamplingTime

Function name: **__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)**

Function description: Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters:
- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
- **SamplingTime:** This parameter can be one of the following values:

  - LL_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_7CYCLES_5
  - LL_ADC_SAMPLINGTIME_13CYCLES_5
  - LL_ADC_SAMPLINGTIME_28CYCLES_5
  - LL_ADC_SAMPLINGTIME_41CYCLES_5
  - LL_ADC_SAMPLINGTIME_55CYCLES_5
  - LL_ADC_SAMPLINGTIME_71CYCLES_5
  - LL_ADC_SAMPLINGTIME_239CYCLES_5

Return values
- **None:**

Notes
- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. Refer to reference manual for ADC processing time of this STM32 serie.
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:
- SMPR1 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP0 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name
__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel)

Function description
Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_7CYCLES_5
  - LL_ADC_SAMPLINGTIME_13CYCLES_5
  - LL_ADC_SAMPLINGTIME_28CYCLES_5
  - LL_ADC_SAMPLINGTIME_41CYCLES_5
  - LL_ADC_SAMPLINGTIME_55CYCLES_5
  - LL_ADC_SAMPLINGTIME_71CYCLES_5
  - LL_ADC_SAMPLINGTIME_239CYCLES_5

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. Refer to reference manual for ADC processing time of this STM32 serie.

Reference Manual to LL API cross reference:

- SMPR1 SMP17 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP16 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP15 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP14 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP13 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP12 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP11 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP10 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP9 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP8 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP7 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP6 LL_ADC_GetChannelSamplingTime

- SMPR2 SMP5 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP4 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP3 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP2 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP1 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP0 LL_ADC_GetChannelSamplingTime

LL_ADC_SetAnalogWDMonitChannels

Function name **__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDChannelGroup)**

Function description Set ADC analog watchdog monitored channels: a single channel or all channels, on ADC groups regular and-or injected.

Parameters
- **ADCx:** ADC instance
- **AWDChannelGroup:** This parameter can be one of the following values: (1) On STM32F1, parameter available only on ADC instance: ADC1.
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG
  - LL_ADC_AWD_ALL_CHANNELS_INJ
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG
  - LL_ADC_AWD_CHANNEL_0_INJ
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG
  - LL_ADC_AWD_CHANNEL_1_INJ
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG
  - LL_ADC_AWD_CHANNEL_2_INJ
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG
  - LL_ADC_AWD_CHANNEL_3_INJ
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG
  - LL_ADC_AWD_CHANNEL_4_INJ
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG
  - LL_ADC_AWD_CHANNEL_5_INJ
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG
  - LL_ADC_AWD_CHANNEL_6_INJ
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG
  - LL_ADC_AWD_CHANNEL_7_INJ
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG
  - LL_ADC_AWD_CHANNEL_8_INJ
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG
  - LL_ADC_AWD_CHANNEL_9_INJ
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG

- LL_ADC_AWD_CHANNEL_10_INJ
- LL_ADC_AWD_CHANNEL_10_REG_INJ
- LL_ADC_AWD_CHANNEL_11_REG
- LL_ADC_AWD_CHANNEL_11_INJ
- LL_ADC_AWD_CHANNEL_11_REG_INJ
- LL_ADC_AWD_CHANNEL_12_REG
- LL_ADC_AWD_CHANNEL_12_INJ
- LL_ADC_AWD_CHANNEL_12_REG_INJ
- LL_ADC_AWD_CHANNEL_13_REG
- LL_ADC_AWD_CHANNEL_13_INJ
- LL_ADC_AWD_CHANNEL_13_REG_INJ
- LL_ADC_AWD_CHANNEL_14_REG
- LL_ADC_AWD_CHANNEL_14_INJ
- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG
- LL_ADC_AWD_CHANNEL_15_INJ
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG
- LL_ADC_AWD_CHANNEL_16_INJ
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG
- LL_ADC_AWD_CHANNEL_17_INJ
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (1)
- LL_ADC_AWD_CH_VREFINT_INJ (1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (1)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (1)

Return values
- **None:**

Notes
- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured).

Reference Manual to LL API cross reference:
- CR1 AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CR1 AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CR1 AWD1EN LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name **__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx)**

Function description

Get ADC analog watchdog monitored channel.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG
  - LL_ADC_AWD_ALL_CHANNELS_INJ
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG
  - LL_ADC_AWD_CHANNEL_0_INJ
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG
  - LL_ADC_AWD_CHANNEL_1_INJ
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG
  - LL_ADC_AWD_CHANNEL_2_INJ
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG
  - LL_ADC_AWD_CHANNEL_3_INJ
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG
  - LL_ADC_AWD_CHANNEL_4_INJ
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG
  - LL_ADC_AWD_CHANNEL_5_INJ
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG
  - LL_ADC_AWD_CHANNEL_6_INJ
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG
  - LL_ADC_AWD_CHANNEL_7_INJ
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG
  - LL_ADC_AWD_CHANNEL_8_INJ
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG
  - LL_ADC_AWD_CHANNEL_9_INJ
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG
  - LL_ADC_AWD_CHANNEL_10_INJ
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG
  - LL_ADC_AWD_CHANNEL_11_INJ
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG
  - LL_ADC_AWD_CHANNEL_12_INJ
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG
  - LL_ADC_AWD_CHANNEL_13_INJ
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG
  - LL_ADC_AWD_CHANNEL_14_INJ

- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG
- LL_ADC_AWD_CHANNEL_15_INJ
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG
- LL_ADC_AWD_CHANNEL_16_INJ
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG
- LL_ADC_AWD_CHANNEL_17_INJ
- LL_ADC_AWD_CHANNEL_17_REG_INJ

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured).

Reference Manual to LL API cross reference:

- CR1 AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CR1 AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CR1 AWD1EN LL_ADC_GetAnalogWDMonitChannels

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- On this STM32 serie, there is only 1 kind of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution:

resolution is not limited (corresponds to ADC resolution configured).

Reference Manual to LL API cross reference:
- HTR HT LL_ADC_SetAnalogWDThresholds
- LTR LT LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name: **__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow)**

Function description: Get ADC analog watchdog threshold value of threshold high or threshold low.

Parameters:
- **ADCx:** ADC instance
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW

Return values:
- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes:
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().

Reference Manual to LL API cross reference:
- HTR HT LL_ADC_GetAnalogWDThresholds
- LTR LT LL_ADC_GetAnalogWDThresholds

LL_ADC_SetMultimode

Function name: **__STATIC_INLINE void LL_ADC_SetMultimode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Multimode)**

Function description: Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters:
- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **Multimode:** This parameter can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL_FAST
  - LL_ADC_MULTI_DUAL_REG_INTERL_SLOW
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INTFAST_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_INTSLOW_INJ_SIM

Return values
- **None:**

Notes
- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:
- CR1 DUALMOD LL_ADC_SetMultimode

LL_ADC_GetMultimode

Function name
__STATIC_INLINE uint32_t LL_ADC_GetMultimode (ADC_Common_TypeDef * ADCxy_COMMON)

Function description
Get ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters
- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values
- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL_FAST
  - LL_ADC_MULTI_DUAL_REG_INTERL_SLOW
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INTFAST_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_INTSLOW_INJ_SIM

Notes
- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:
- CR1 DUALMOD LL_ADC_GetMultimode

LL_ADC_Enable

Function name
__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description
Enable the selected ADC instance.

Parameters
- **ADCx:** ADC instance

Return values
- **None:**

Notes
- On this STM32 serie, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.

Reference Manual to LL API cross
- CR2 ADON LL_ADC_Enable

reference:

LL_ADC_Disable

Function name	**__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)**
Function description	Disable the selected ADC instance.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 ADON LL_ADC_Disable

LL_ADC_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)**
Function description	Get the selected ADC instance enable state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** ADC is disabled, 1: ADC is enabled.
Reference Manual to LL API cross reference:	• CR2 ADON LL_ADC_IsEnabled

LL_ADC_StartCalibration

Function name	**__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx)**
Function description	Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, before starting a calibration, ADC must be disabled. A minimum number of ADC clock cycles are required between ADC disable state and calibration start. Refer to literal LL_ADC_DELAY_DISABLE_CALIB_ADC_CYCLES. • On this STM32 serie, hardware prerequisite before starting a calibration: the ADC must have been in power-on state for at least two ADC clock cycles.
Reference Manual to LL API cross reference:	• CR2 CAL LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing**

(ADC_TypeDef * ADCx)

Function description	Get ADC calibration state.
Parameters	• **ADCx:** ADC instance
Return values	• **0:** calibration complete, 1: calibration in progress.
Reference Manual to LL API cross reference:	• CR2 CAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversionSWStart

Function name	**__STATIC_INLINE void LL_ADC_REG_StartConversionSWStart (ADC_TypeDef * ADCx)**
Function description	Start ADC group regular conversion.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Notes	• On this STM32 serie, this function is relevant only for internal trigger (SW start), not for external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion start must be performed using function LL_ADC_REG_StartConversionExtTrig(). (if external trigger edge would have been set during ADC other settings, ADC conversion would start at trigger event as soon as ADC is enabled).
Reference Manual to LL API cross reference:	• CR2 SWSTART LL_ADC_REG_StartConversionSWStart

LL_ADC_REG_StartConversionExtTrig

Function name	**__STATIC_INLINE void LL_ADC_REG_StartConversionExtTrig (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)**
Function description	Start ADC group regular conversion from external trigger.
Parameters	• **ExternalTriggerEdge:** This parameter can be one of the following values: – LL_ADC_REG_TRIG_EXT_RISING • **ADCx:** ADC instance
Return values	• **None:**
Notes	• ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command. • On this STM32 serie, this function is relevant for ADC conversion start from external trigger. If internal trigger (SW start) is needed, perform ADC conversion start using function LL_ADC_REG_StartConversionSWStart().
Reference Manual to	• CR2 EXTEN LL_ADC_REG_StartConversionExtTrig

LL API cross reference:

LL_ADC_REG_StopConversionExtTrig

Function name **__STATIC_INLINE void LL_ADC_REG_StopConversionExtTrig (ADC_TypeDef * ADCx)**

Function description Stop ADC group regular conversion from external trigger.

Parameters
- **ADCx:** ADC instance

Return values
- **None:**

Notes
- No more ADC conversion will start at next trigger event following the ADC stop conversion command. If a conversion is on-going, it will be completed.
- On this STM32 serie, there is no specific command to stop a conversion on-going or to stop ADC converting in continuous mode. These actions can be performed using function LL_ADC_Disable().

Reference Manual to LL API cross reference:
- CR2 EXTSEL LL_ADC_REG_StopConversionExtTrig

LL_ADC_REG_ReadConversionData32

Function name **__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (ADC_TypeDef * ADCx)**

Function description Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters
- **ADCx:** ADC instance

Return values
- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:
- DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData12

Function name **__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (ADC_TypeDef * ADCx)**

Function description Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters
- **ADCx:** ADC instance

Return values
- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes
- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be

needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadMultiConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConversionData32 (ADC_TypeDef * ADCx, uint32_t ConversionData)

Function description

Get ADC multimode conversion data of ADC master, ADC slave or raw data with ADC master and slave concatenated.

Parameters

- **ADCx:** ADC instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ConversionData:** This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
  - LL_ADC_MULTI_MASTER_SLAVE

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- If raw data with ADC master and slave concatenated is retrieved, a macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE(). (however this macro is mainly intended for multimode transfer by DMA, because this function can do the same by getting multimode conversion data of ADC master or ADC slave separately).

Reference Manual to LL API cross reference:

- DR DATA LL_ADC_REG_ReadMultiConversionData32
- DR ADC2DATA LL_ADC_REG_ReadMultiConversionData32

LL_ADC_INJ_StartConversionSWStart

Function name

__STATIC_INLINE void LL_ADC_INJ_StartConversionSWStart (ADC_TypeDef * ADCx)

Function description

Start ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, this function is relevant only for internal trigger (SW start), not for external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion start must be performed using function LL_ADC_INJ_StartConversionExtTrig(). (if external trigger edge would have been set during ADC other settings, ADC conversion would start at trigger event as soon as ADC is

enabled).

Reference Manual to LL API cross reference:
- CR2 JSWSTART LL_ADC_INJ_StartConversionSWStart

LL_ADC_INJ_StartConversionExtTrig

Function name **__STATIC_INLINE void LL_ADC_INJ_StartConversionExtTrig (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)**

Function description Start ADC group injected conversion from external trigger.

Parameters
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  – LL_ADC_INJ_TRIG_EXT_RISING
- **ADCx:** ADC instance

Return values
- **None:**

Notes
- ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 serie, this function is relevant for ADC conversion start from external trigger. If internal trigger (SW start) is needed, perform ADC conversion start using function LL_ADC_INJ_StartConversionSWStart().

Reference Manual to LL API cross reference:
- CR2 JEXTEN LL_ADC_INJ_StartConversionExtTrig

LL_ADC_INJ_StopConversionExtTrig

Function name **__STATIC_INLINE void LL_ADC_INJ_StopConversionExtTrig (ADC_TypeDef * ADCx)**

Function description Stop ADC group injected conversion from external trigger.

Parameters
- **ADCx:** ADC instance

Return values
- **None:**

Notes
- No more ADC conversion will start at next trigger event following the ADC stop conversion command. If a conversion is on-going, it will be completed.
- On this STM32 serie, there is no specific command to stop a conversion on-going or to stop ADC converting in continuous mode. These actions can be performed using function LL_ADC_Disable().

Reference Manual to LL API cross reference:
- CR2 JEXTSEL LL_ADC_INJ_StopConversionExtTrig

LL_ADC_INJ_ReadConversionData32

Function name **__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (ADC_TypeDef * ADCx,**

uint32_t Rank)

Function description: Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters:
- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values:
- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:
- JDR1 JDATA LL_ADC_INJ_ReadConversionData32
- JDR2 JDATA LL_ADC_INJ_ReadConversionData32
- JDR3 JDATA LL_ADC_INJ_ReadConversionData32
- JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData12

Function name: **__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (ADC_TypeDef * ADCx, uint32_t Rank)**

Function description: Get ADC group injected conversion data, range fit for ADC resolution 12 bits.

Parameters:
- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values:
- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes:
- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:
- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_IsActiveFlag_EOS

Function name: **__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (ADC_TypeDef * ADCx)**

Function description: Get flag ADC group regular end of sequence conversions.

Parameters:
- **ADCx:** ADC instance

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• SR EOC LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_JEOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (ADC_TypeDef * ADCx)**
Function description	Get flag ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR JEOC LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_AWD1

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (ADC_TypeDef * ADCx)**
Function description	Get flag ADC analog watchdog 1 flag.
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR AWD LL_ADC_IsActiveFlag_AWD1

LL_ADC_ClearFlag_EOS

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR EOC LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_JEOS

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross	• SR JEOC LL_ADC_ClearFlag_JEOS

reference:

LL_ADC_ClearFlag_AWD1

Function name	**__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)**
Function description	Clear flag ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR AWD LL_ADC_ClearFlag_AWD1

LL_ADC_IsActiveFlag_MST_EOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOS (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get flag multimode ADC group regular end of sequence conversions of the ADC master.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR EOC LL_ADC_IsActiveFlag_MST_EOS

LL_ADC_IsActiveFlag_SLV_EOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOS (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get flag multimode ADC group regular end of sequence conversions of the ADC slave.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR EOC LL_ADC_IsActiveFlag_SLV_EOS

LL_ADC_IsActiveFlag_MST_JEOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOS (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get flag multimode ADC group injected end of sequence conversions of the ADC master.

Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR JEOC LL_ADC_IsActiveFlag_MST_JEOS

LL_ADC_IsActiveFlag_SLV_JEOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOS (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get flag multimode ADC group injected end of sequence conversions of the ADC slave.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR JEOC LL_ADC_IsActiveFlag_SLV_JEOS

LL_ADC_IsActiveFlag_MST_AWD1

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD1 (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get flag multimode ADC analog watchdog 1 of the ADC master.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR AWD LL_ADC_IsActiveFlag_MST_AWD1

LL_ADC_IsActiveFlag_SLV_AWD1

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD1 (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	Get flag multimode analog watchdog 1 of the ADC slave.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR AWD LL_ADC_IsActiveFlag_SLV_AWD1

LL_ADC_EnableIT_EOS

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 EOCIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_JEOS

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 JEOCIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_AWD1

Function name	**__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Enable interruption ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 AWDIE LL_ADC_EnableIT_AWD1

LL_ADC_DisableIT_EOS

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group regular end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 EOCIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_JEOS

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC group injected end of sequence conversions.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 JEOCIE LL_ADC_EnableIT_JEOS

LL_ADC_DisableIT_AWD1

Function name	**__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Disable interruption ADC analog watchdog 1.
Parameters	• **ADCx:** ADC instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 AWDIE LL_ADC_EnableIT_AWD1

LL_ADC_IsEnabledIT_EOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 EOCIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_JEOS

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 JEOCIE LL_ADC_EnableIT_JEOS

LL_ADC_IsEnabledIT_AWD1

Function name	**__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (ADC_TypeDef * ADCx)**
Function description	Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).
Parameters	• **ADCx:** ADC instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 AWDIE LL_ADC_EnableIT_AWD1

LL_ADC_CommonDeInit

Function name	**ErrorStatus LL_ADC_CommonDeInit (ADC_Common_TypeDef * ADCxy_COMMON)**
Function description	De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC common registers are de-initialized – ERROR: not applicable

LL_ADC_CommonInit

Function name	**ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)**
Function description	Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).
Parameters	• **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE()) • **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: ADC common registers are initialized – ERROR: ADC common registers are not initialized
Notes	• The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name	**void LL_ADC_CommonStructInit**

(LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)

Function description: Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters:
- **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.

Return values:
- **None:**

LL_ADC_DeInit

Function name: **ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)**

Function description: De-initialize registers of the selected ADC instance to their default reset values.

Parameters:
- **ADCx:** ADC instance

Return values:
- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are de-initialized
  - ERROR: ADC registers are not de-initialized

Notes:
- To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name: **ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)**

Function description: Initialize some features of ADC instance.

Parameters:
- **ADCx:** ADC instance
- **ADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values:
- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes:
- These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 families). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, some other features must be configured using LL unitary functions. The minimum

configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name **void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)**

Function description Set each LL_ADC_InitTypeDef field to default value.

Parameters
- **ADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.

Return values
- **None:**

LL_ADC_REG_Init

Function name **ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)**

Function description Initialize some features of ADC group regular.

Parameters
- **ADCx:** ADC instance
- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: ADC registers are initialized
  – ERROR: ADC registers are not initialized

Notes
- These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name **void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)**

Function description Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters
- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values
- **None:**

LL_ADC_INJ_Init

Function name **ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * ADCx, LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)**

Function description Initialize some features of ADC group injected.

Parameters
- **ADCx:** ADC instance
- **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes
- These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_INJ_StructInit

Function name **void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)**

Function description Set each LL_ADC_INJ_InitTypeDef field to default value.

Parameters
- **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef

structure whose fields will be set to default values.

Return values

- **None:**

46.3 ADC Firmware driver defines

46.3.1 ADC

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE	ADC analog watchdog monitoring disabled
LL_ADC_AWD_ALL_CHANNELS_REG	ADC analog watchdog monitoring of all channels, converted by group regular only
LL_ADC_AWD_ALL_CHANNELS_INJ	ADC analog watchdog monitoring of all channels, converted by group injected only
LL_ADC_AWD_ALL_CHANNELS_REG_INJ	ADC analog watchdog monitoring of all channels, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_0_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group regular only
LL_ADC_AWD_CHANNEL_0_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group injected only
LL_ADC_AWD_CHANNEL_0_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_1_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group regular only
LL_ADC_AWD_CHANNEL_1_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group injected only
LL_ADC_AWD_CHANNEL_1_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_2_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2,

	converted by group regular only
LL_ADC_AWD_CHANNEL_2_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group injected only
LL_ADC_AWD_CHANNEL_2_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_3_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group regular only
LL_ADC_AWD_CHANNEL_3_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group injected only
LL_ADC_AWD_CHANNEL_3_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_4_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group regular only
LL_ADC_AWD_CHANNEL_4_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group injected only
LL_ADC_AWD_CHANNEL_4_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_5_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group regular only
LL_ADC_AWD_CHANNEL_5_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group injected only
LL_ADC_AWD_CHANNEL_5_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_6_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group regular only
LL_ADC_AWD_CHANNEL_6_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group injected only
LL_ADC_AWD_CHANNEL_6_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_7_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group regular only
LL_ADC_AWD_CHANNEL_7_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group injected only
LL_ADC_AWD_CHANNEL_7_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_8_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group regular only
LL_ADC_AWD_CHANNEL_8_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group injected only
LL_ADC_AWD_CHANNEL_8_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_9_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group regular only
LL_ADC_AWD_CHANNEL_9_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group injected only
LL_ADC_AWD_CHANNEL_9_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9,

	converted by either group regular or injected
LL_ADC_AWD_CHANNEL_10_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group regular only
LL_ADC_AWD_CHANNEL_10_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group injected only
LL_ADC_AWD_CHANNEL_10_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_11_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group regular only
LL_ADC_AWD_CHANNEL_11_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group injected only
LL_ADC_AWD_CHANNEL_11_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_12_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group regular only
LL_ADC_AWD_CHANNEL_12_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group injected only
LL_ADC_AWD_CHANNEL_12_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_13_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group regular only
LL_ADC_AWD_CHANNEL_13_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group injected only

LL_ADC_AWD_CHANNEL_13_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_14_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group regular only
LL_ADC_AWD_CHANNEL_14_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group injected only
LL_ADC_AWD_CHANNEL_14_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_15_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group regular only
LL_ADC_AWD_CHANNEL_15_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group injected only
LL_ADC_AWD_CHANNEL_15_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_16_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group regular only
LL_ADC_AWD_CHANNEL_16_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group injected only
LL_ADC_AWD_CHANNEL_16_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by either group regular or injected
LL_ADC_AWD_CHANNEL_17_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only
LL_ADC_AWD_CHANNEL_17_INJ	ADC analog watchdog monitoring of ADC external channel (channel

	connected to GPIO pin) ADCx_IN17, converted by group injected only
LL_ADC_AWD_CHANNEL_17_REG_INJ	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by either group regular or injected
LL_ADC_AWD_CH_VREFINT_REG	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only
LL_ADC_AWD_CH_VREFINT_INJ	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only
LL_ADC_AWD_CH_VREFINT_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected
LL_ADC_AWD_CH_TEMPSENSOR_REG	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only
LL_ADC_AWD_CH_TEMPSENSOR_INJ	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group injected only
LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by either group regular or injected

Analog watchdog - Analog watchdog number

LL_ADC_AWD1	ADC analog watchdog number 1

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH	ADC analog watchdog threshold high
LL_ADC_AWD_THRESHOLD_LOW	ADC analog watchdog threshold low

ADC instance - Channel number

LL_ADC_CHANNEL_0	ADC external channel (channel connected to GPIO pin) ADCx_IN0
LL_ADC_CHANNEL_1	ADC external channel (channel connected to GPIO pin) ADCx_IN1
LL_ADC_CHANNEL_2	ADC external channel (channel connected to GPIO pin) ADCx_IN2
LL_ADC_CHANNEL_3	ADC external channel (channel connected to GPIO pin) ADCx_IN3

LL_ADC_CHANNEL_4	ADC external channel (channel connected to GPIO pin) ADCx_IN4
LL_ADC_CHANNEL_5	ADC external channel (channel connected to GPIO pin) ADCx_IN5
LL_ADC_CHANNEL_6	ADC external channel (channel connected to GPIO pin) ADCx_IN6
LL_ADC_CHANNEL_7	ADC external channel (channel connected to GPIO pin) ADCx_IN7
LL_ADC_CHANNEL_8	ADC external channel (channel connected to GPIO pin) ADCx_IN8
LL_ADC_CHANNEL_9	ADC external channel (channel connected to GPIO pin) ADCx_IN9
LL_ADC_CHANNEL_10	ADC external channel (channel connected to GPIO pin) ADCx_IN10
LL_ADC_CHANNEL_11	ADC external channel (channel connected to GPIO pin) ADCx_IN11
LL_ADC_CHANNEL_12	ADC external channel (channel connected to GPIO pin) ADCx_IN12
LL_ADC_CHANNEL_13	ADC external channel (channel connected to GPIO pin) ADCx_IN13
LL_ADC_CHANNEL_14	ADC external channel (channel connected to GPIO pin) ADCx_IN14
LL_ADC_CHANNEL_15	ADC external channel (channel connected to GPIO pin) ADCx_IN15
LL_ADC_CHANNEL_16	ADC external channel (channel connected to GPIO pin) ADCx_IN16
LL_ADC_CHANNEL_17	ADC external channel (channel connected to GPIO pin) ADCx_IN17
LL_ADC_CHANNEL_VREFINT	ADC internal channel connected to VrefInt: Internal voltage reference. On STM32F1, ADC channel available only on ADC instance: ADC1.
LL_ADC_CHANNEL_TEMPSENSOR	ADC internal channel connected to Temperature sensor.

Channel - Sampling time

LL_ADC_SAMPLINGTIME_1CYCLE_5	Sampling time 1.5 ADC clock cycle
LL_ADC_SAMPLINGTIME_7CYCLES_5	Sampling time 7.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_13CYCLES_5	Sampling time 13.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_28CYCLES_5	Sampling time 28.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_41CYCLES_5	Sampling time 41.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_55CYCLES_5	Sampling time 55.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_71CYCLES_5	Sampling time 71.5 ADC clock cycles
LL_ADC_SAMPLINGTIME_239CYCLES_5	Sampling time 239.5 ADC clock cycles

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE	ADC measurement pathes all disabled
LL_ADC_PATH_INTERNAL_VREFINT	ADC measurement path to internal channel VrefInt
LL_ADC_PATH_INTERNAL_TEMPSENSOR	ADC measurement path to internal channel temperature sensor

ADC instance - Data alignment

LL_ADC_DATA_ALIGN_RIGHT	ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)
LL_ADC_DATA_ALIGN_LEFT	ADC conversion data alignment: left aligned (aligment on data register MSB bit 15)

ADC flags

LL_ADC_FLAG_STRT	ADC flag ADC group regular conversion start
LL_ADC_FLAG_EOS	ADC flag ADC group regular end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group regular end of unitary conversion. Flag noted as "EOC" is corresponding to flag "EOS" in other STM32 families)
LL_ADC_FLAG_JSTRT	ADC flag ADC group injected conversion start
LL_ADC_FLAG_JEOS	ADC flag ADC group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)
LL_ADC_FLAG_AWD1	ADC flag ADC analog watchdog 1
LL_ADC_FLAG_EOS_MST	ADC flag ADC multimode master group regular end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group regular end of unitary conversion. Flag noted as "EOC" is corresponding to flag "EOS" in other STM32 families)
LL_ADC_FLAG_EOS_SLV	ADC flag ADC multimode slave group regular end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group regular end of unitary conversion. Flag noted as "EOC" is corresponding to flag "EOS" in other STM32 families) (on STM32F1, this flag must be read from ADC instance slave: ADC2)
LL_ADC_FLAG_JEOS_MST	ADC flag ADC multimode master group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)
LL_ADC_FLAG_JEOS_SLV	ADC flag ADC multimode slave group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families) (on STM32F1, this flag must be read from ADC instance slave: ADC2)
LL_ADC_FLAG_AWD1_MST	ADC flag ADC multimode master analog watchdog 1 of the ADC master

LL_ADC_FLAG_AWD1_SLV	ADC flag ADC multimode slave analog watchdog 1 of the ADC slave (on STM32F1, this flag must be read from ADC instance slave: ADC2)

ADC instance - Groups

LL_ADC_GROUP_REGULAR	ADC group regular (available on all STM32 devices)
LL_ADC_GROUP_INJECTED	ADC group injected (not available on all STM32 devices)
LL_ADC_GROUP_REGULAR_INJECTED	ADC both groups regular and injected

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_TEMPSENSOR_STAB_US	Delay for internal voltage reference stabilization time
LL_ADC_DELAY_DISABLE_CALIB_ADC_CYCLES	Delay required between ADC disable and ADC calibration start
LL_ADC_DELAY_ENABLE_CALIB_ADC_CYCLES	Delay required between end of ADC enable and the start of ADC calibration

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE	ADC group injected sequencer discontinuous mode disable
LL_ADC_INJ_SEQ_DISCONT_1RANK	ADC group injected sequencer discontinuous mode enable with sequence interruption every rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1	ADC group injected sequencer rank 1
LL_ADC_INJ_RANK_2	ADC group injected sequencer rank 2
LL_ADC_INJ_RANK_3	ADC group injected sequencer rank 3
LL_ADC_INJ_RANK_4	ADC group injected sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE	ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)
LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS	ADC group injected sequencer enable with 2 ranks in the sequence
LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS	ADC group injected sequencer enable with 3 ranks in the sequence
LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS	ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING	ADC group injected conversion trigger polarity set to rising edge

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE	ADC group injected conversion trigger internal: SW start.
LL_ADC_INJ_TRIG_EXT_TIM1_TRGO	ADC group injected conversion trigger from external IP: TIM1 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM1_CH4	ADC group injected conversion trigger from external IP: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM2_TRGO	ADC group injected conversion trigger from external IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM2_CH1	ADC group injected conversion trigger from external IP: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM3_CH4	ADC group injected conversion trigger from external IP: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM4_TRGO	ADC group injected conversion trigger from external IP: TIM4 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_EXTI_LINE15	ADC group injected conversion trigger from external IP: external interrupt line 15. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM8_CH4	ADC group injected conversion trigger from external IP: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Available only on high-density and XL-density devices. A remap of trigger must be done at top level (refer to AFIO peripheral).
LL_ADC_INJ_TRIG_EXT_TIM4_CH3	ADC group injected conversion trigger from external IP: TIM4 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM8_CH2	ADC group injected conversion trigger from external IP: TIM8 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM8_CH4_ADC3	ADC group injected conversion trigger from external IP: TIM8 channel 4 event (capture

compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM5_TRGO	ADC group injected conversion trigger from external IP: TIM5 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_INJ_TRIG_EXT_TIM5_CH4	ADC group injected conversion trigger from external IP: TIM5 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT	ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.
LL_ADC_INJ_TRIG_FROM_GRP_REGULAR	ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_EOS	ADC interruption ADC group regular end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group regular end of unitary conversion. Flag noted as "EOC" is corresponding to flag "EOS" in other STM32 families)
LL_ADC_IT_JEOS	ADC interruption ADC group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)
LL_ADC_IT_AWD1	ADC interruption ADC analog watchdog 1

Multimode - ADC master or slave

LL_ADC_MULTI_MASTER	In multimode, selection among several ADC instances: ADC master
LL_ADC_MULTI_SLAVE	In multimode, selection among several ADC instances: ADC slave
LL_ADC_MULTI_MASTER_SLAVE	In multimode, selection among several ADC instances: both ADC master and ADC slave

Multimode - Mode

LL_ADC_MULTI_INDEPENDENT	ADC dual mode disabled (ADC independent mode)
LL_ADC_MULTI_DUAL_REG_SIMULT	ADC dual mode enabled: group regular simultaneous
LL_ADC_MULTI_DUAL_REG_INTERL_FA	ADC dual mode enabled: Combined group regular interleaved fast (delay between

ST	ADC sampling phases: 7 ADC clock cycles) (equivalent to multimode sampling delay set to "LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES" on other STM32 devices))
LL_ADC_MULTI_DUAL_REG_INTERL_SLOW	ADC dual mode enabled: Combined group regular interleaved slow (delay between ADC sampling phases: 14 ADC clock cycles) (equivalent to multimode sampling delay set to "LL_ADC_MULTI_TWOSMP_DELAY_14CYCLES" on other STM32 devices))
LL_ADC_MULTI_DUAL_INJ_SIMULT	ADC dual mode enabled: group injected simultaneous slow (delay between ADC sampling phases: 14 ADC clock cycles) (equivalent to multimode sampling delay set to "LL_ADC_MULTI_TWOSMP_DELAY_14CYCLES" on other STM32 devices))
LL_ADC_MULTI_DUAL_INJ_ALTERN	ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)
LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM	ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous
LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT	ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger
LL_ADC_MULTI_DUAL_REG_INTFAST_INJ_SIM	ADC dual mode enabled: Combined group regular interleaved fast (delay between ADC sampling phases: 7 ADC clock cycles) + group injected simultaneous
LL_ADC_MULTI_DUAL_REG_INTSLOW_INJ_SIM	ADC dual mode enabled: Combined group regular interleaved slow (delay between ADC sampling phases: 14 ADC clock cycles) + group injected simultaneous

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

LL_ADC_DMA_REG_REGULAR_DATA_MULTI

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE	ADC conversions are performed in single mode: one conversion per trigger
LL_ADC_REG_CONV_CONTINUOUS	ADC conversions are performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE	ADC conversions are not transferred by

	DMA
LL_ADC_REG_DMA_TRANSFER_UNLIMITED	ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE	ADC group regular sequencer discontinuous mode disable
LL_ADC_REG_SEQ_DISCONT_1RANK	ADC group regular sequencer discontinuous mode enable with sequence interruption every rank
LL_ADC_REG_SEQ_DISCONT_2RANKS	ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks
LL_ADC_REG_SEQ_DISCONT_3RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks
LL_ADC_REG_SEQ_DISCONT_4RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks
LL_ADC_REG_SEQ_DISCONT_5RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks
LL_ADC_REG_SEQ_DISCONT_6RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks
LL_ADC_REG_SEQ_DISCONT_7RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks
LL_ADC_REG_SEQ_DISCONT_8RANKS	ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1	ADC group regular sequencer rank 1
LL_ADC_REG_RANK_2	ADC group regular sequencer rank 2
LL_ADC_REG_RANK_3	ADC group regular sequencer rank 3
LL_ADC_REG_RANK_4	ADC group regular sequencer rank 4
LL_ADC_REG_RANK_5	ADC group regular sequencer rank 5
LL_ADC_REG_RANK_6	ADC group regular sequencer rank 6
LL_ADC_REG_RANK_7	ADC group regular sequencer rank 7
LL_ADC_REG_RANK_8	ADC group regular sequencer rank 8

LL_ADC_REG_RANK_9	ADC group regular sequencer rank 9
LL_ADC_REG_RANK_10	ADC group regular sequencer rank 10
LL_ADC_REG_RANK_11	ADC group regular sequencer rank 11
LL_ADC_REG_RANK_12	ADC group regular sequencer rank 12
LL_ADC_REG_RANK_13	ADC group regular sequencer rank 13
LL_ADC_REG_RANK_14	ADC group regular sequencer rank 14
LL_ADC_REG_RANK_15	ADC group regular sequencer rank 15
LL_ADC_REG_RANK_16	ADC group regular sequencer rank 16

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE	ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)
LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS	ADC group regular sequencer enable with 2 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS	ADC group regular sequencer enable with 3 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS	ADC group regular sequencer enable with 4 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS	ADC group regular sequencer enable with 5 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS	ADC group regular sequencer enable with 6 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS	ADC group regular sequencer enable with 7 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS	ADC group regular sequencer enable with 8 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS	ADC group regular sequencer enable with 9 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS	ADC group regular sequencer enable with 10 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS	ADC group regular sequencer enable with 11 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS	ADC group regular sequencer enable with 12 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS	ADC group regular sequencer enable with 13 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS	ADC group regular sequencer enable with 14 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS	ADC group regular sequencer enable with 15 ranks in the sequence
LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS	ADC group regular sequencer enable with 16 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING	ADC group regular conversion trigger polarity set to rising edge

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE	ADC group regular conversion trigger internal: SW start.
LL_ADC_REG_TRIG_EXT_TIM1_CH3	ADC group regular conversion trigger from external IP: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM1_CH1	ADC group regular conversion trigger from external IP: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM1_CH2	ADC group regular conversion trigger from external IP: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM2_CH2	ADC group regular conversion trigger from external IP: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM3_TRGO	ADC group regular conversion trigger from external IP: TIM3 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM4_CH4	ADC group regular conversion trigger from external IP: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_EXTI_LINE11	ADC group regular conversion trigger from external IP: external interrupt line 11. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM8_TRGO	ADC group regular conversion trigger from external IP: TIM8 TRGO. Trigger edge set to rising edge (default setting). Available only on high-density and XL-density devices. A remap of trigger must be done at top level (refer to AFIO peripheral).
LL_ADC_REG_TRIG_EXT_TIM3_CH1	ADC group regular conversion trigger from external IP: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_CH3	ADC group regular conversion trigger from external IP: TIM2 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM8_CH1	ADC group regular conversion trigger from external IP: TIM8 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM8_TRGO_ADC3	ADC group regular conversion trigger from external IP: TIM8 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM5_CH1	ADC group regular conversion trigger from external IP: TIM5 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM5_CH3	ADC group regular conversion trigger from external IP: TIM5 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_12B ADC resolution 12 bits

ADC instance - Scan selection

LL_ADC_SEQ_SCAN_DISABLE	ADC conversion is performed in unitary conversion mode (one channel converted, that defined in rank 1). Configuration of both groups regular and injected sequencers (sequence length, ...) is discarded: equivalent to length of 1 rank.
LL_ADC_SEQ_SCAN_ENABLE	ADC conversions are performed in sequence conversions mode, according to configuration of both groups regular and injected sequencers (sequence length, ...).

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4

  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12

  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel:

LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0

  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition

of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG
  - LL_ADC_AWD_ALL_CHANNELS_INJ
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG

- LL_ADC_AWD_CHANNEL_0_INJ
- LL_ADC_AWD_CHANNEL_0_REG_INJ
- LL_ADC_AWD_CHANNEL_1_REG
- LL_ADC_AWD_CHANNEL_1_INJ
- LL_ADC_AWD_CHANNEL_1_REG_INJ
- LL_ADC_AWD_CHANNEL_2_REG
- LL_ADC_AWD_CHANNEL_2_INJ
- LL_ADC_AWD_CHANNEL_2_REG_INJ
- LL_ADC_AWD_CHANNEL_3_REG
- LL_ADC_AWD_CHANNEL_3_INJ
- LL_ADC_AWD_CHANNEL_3_REG_INJ
- LL_ADC_AWD_CHANNEL_4_REG
- LL_ADC_AWD_CHANNEL_4_INJ
- LL_ADC_AWD_CHANNEL_4_REG_INJ
- LL_ADC_AWD_CHANNEL_5_REG
- LL_ADC_AWD_CHANNEL_5_INJ
- LL_ADC_AWD_CHANNEL_5_REG_INJ
- LL_ADC_AWD_CHANNEL_6_REG
- LL_ADC_AWD_CHANNEL_6_INJ
- LL_ADC_AWD_CHANNEL_6_REG_INJ
- LL_ADC_AWD_CHANNEL_7_REG
- LL_ADC_AWD_CHANNEL_7_INJ
- LL_ADC_AWD_CHANNEL_7_REG_INJ
- LL_ADC_AWD_CHANNEL_8_REG
- LL_ADC_AWD_CHANNEL_8_INJ
- LL_ADC_AWD_CHANNEL_8_REG_INJ
- LL_ADC_AWD_CHANNEL_9_REG
- LL_ADC_AWD_CHANNEL_9_INJ
- LL_ADC_AWD_CHANNEL_9_REG_INJ
- LL_ADC_AWD_CHANNEL_10_REG
- LL_ADC_AWD_CHANNEL_10_INJ
- LL_ADC_AWD_CHANNEL_10_REG_INJ
- LL_ADC_AWD_CHANNEL_11_REG
- LL_ADC_AWD_CHANNEL_11_INJ
- LL_ADC_AWD_CHANNEL_11_REG_INJ
- LL_ADC_AWD_CHANNEL_12_REG
- LL_ADC_AWD_CHANNEL_12_INJ
- LL_ADC_AWD_CHANNEL_12_REG_INJ
- LL_ADC_AWD_CHANNEL_13_REG
- LL_ADC_AWD_CHANNEL_13_INJ
- LL_ADC_AWD_CHANNEL_13_REG_INJ
- LL_ADC_AWD_CHANNEL_14_REG
- LL_ADC_AWD_CHANNEL_14_INJ
- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG
- LL_ADC_AWD_CHANNEL_15_INJ
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG
- LL_ADC_AWD_CHANNEL_16_INJ
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG
- LL_ADC_AWD_CHANNEL_17_INJ

- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (1)
- LL_ADC_AWD_CH_VREFINT_INJ (1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (1)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (1)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example: LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1, __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
- __AWD_THRESHOLD__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param>, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is

different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
- __AWD_THRESHOLD_12_BITS__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_6_bits> = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled. In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common

instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter. On STM32F1, there is no common ADC instance. However, ADC instance ADC1 has a role of common ADC instance for ADC1 and ADC2: this instance is used to manage internal channels and multimode (these features are managed in ADC common instances on some other STM32 devices). ADC instance ADC3 (if available on the selected device) has no ADC common instance.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances). On STM32F1, there is no common ADC instance. However, ADC instance ADC1 has a role of common ADC instance for ADC1 and ADC2: this instance is used to manage internal

channels and multimode (these features are managed in ADC common instances on some other STM32 devices). ADC instance ADC3 (if available on the selected device) has no ADC common instance.

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  – LL_ADC_RESOLUTION_12B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  – LL_ADC_RESOLUTION_12B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be known from user board environment or can be calculated using ADC measurement.

__LL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32F1, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32F1, refer to device datasheet parameter "V25".
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  – LL_ADC_RESOLUTION_12B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be known from user board

environment or can be calculated using ADC measurement. ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

47 LL BUS Generic Driver

47.1 BUS Firmware driver API description

47.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableClock

Function name **__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)**

Function description Enable AHB1 peripherals clock.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2 (*)
  - LL_AHB1_GRP1_PERIPH_ETHMAC (*)
  - LL_AHB1_GRP1_PERIPH_ETHMACRX (*)
  - LL_AHB1_GRP1_PERIPH_ETHMACTX (*)
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_FSMC (*)
  - LL_AHB1_GRP1_PERIPH_OTGFS (*)
  - LL_AHB1_GRP1_PERIPH_SDIO (*)
  - LL_AHB1_GRP1_PERIPH_SRAM

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHBENR CRCEN LL_AHB1_GRP1_EnableClock
- AHBENR DMA1EN LL_AHB1_GRP1_EnableClock
- AHBENR DMA2EN LL_AHB1_GRP1_EnableClock
- AHBENR ETHMACEN LL_AHB1_GRP1_EnableClock
- AHBENR ETHMACRXEN LL_AHB1_GRP1_EnableClock
- AHBENR ETHMACTXEN LL_AHB1_GRP1_EnableClock
- AHBENR FLITFEN LL_AHB1_GRP1_EnableClock
- AHBENR FSMCEN LL_AHB1_GRP1_EnableClock
- AHBENR OTGFSEN LL_AHB1_GRP1_EnableClock
- AHBENR SDIOEN LL_AHB1_GRP1_EnableClock
- AHBENR SRAMEN LL_AHB1_GRP1_EnableClock

LL_AHB1_GRP1_IsEnabledClock

Function name **__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)**

Function description Check if AHB1 peripheral clock is enabled or not.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2 (*)
  - LL_AHB1_GRP1_PERIPH_ETHMAC (*)

- LL_AHB1_GRP1_PERIPH_ETHMACRX (*)
- LL_AHB1_GRP1_PERIPH_ETHMACTX (*)
- LL_AHB1_GRP1_PERIPH_FLASH
- LL_AHB1_GRP1_PERIPH_FSMC (*)
- LL_AHB1_GRP1_PERIPH_OTGFS (*)
- LL_AHB1_GRP1_PERIPH_SDIO (*)
- LL_AHB1_GRP1_PERIPH_SRAM

Return values
- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:
- AHBENR CRCEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR DMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHBENR DMA2EN LL_AHB1_GRP1_IsEnabledClock
- AHBENR ETHMACEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR ETHMACRXEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR ETHMACTXEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR FLITFEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR FSMCEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR OTGFSEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR SDIOEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR SRAMEN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name
__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description
Disable AHB1 peripherals clock.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2 (*)
  - LL_AHB1_GRP1_PERIPH_ETHMAC (*)
  - LL_AHB1_GRP1_PERIPH_ETHMACRX (*)
  - LL_AHB1_GRP1_PERIPH_ETHMACTX (*)
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_FSMC (*)
  - LL_AHB1_GRP1_PERIPH_OTGFS (*)
  - LL_AHB1_GRP1_PERIPH_SDIO (*)
  - LL_AHB1_GRP1_PERIPH_SRAM

Return values
- **None:**

Reference Manual to LL API cross reference:
- AHBENR CRCEN LL_AHB1_GRP1_DisableClock
- AHBENR DMA1EN LL_AHB1_GRP1_DisableClock
- AHBENR DMA2EN LL_AHB1_GRP1_DisableClock
- AHBENR ETHMACEN LL_AHB1_GRP1_DisableClock
- AHBENR ETHMACRXEN LL_AHB1_GRP1_DisableClock
- AHBENR ETHMACTXEN LL_AHB1_GRP1_DisableClock
- AHBENR FLITFEN LL_AHB1_GRP1_DisableClock
- AHBENR FSMCEN LL_AHB1_GRP1_DisableClock
- AHBENR OTGFSEN LL_AHB1_GRP1_DisableClock
- AHBENR SDIOEN LL_AHB1_GRP1_DisableClock

- AHBENR SRAMEN LL_AHB1_GRP1_DisableClock

LL_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP1_PERIPH_BKP
  - LL_APB1_GRP1_PERIPH_CAN1 (*)
  - LL_APB1_GRP1_PERIPH_CAN2 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_DAC1 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_SPI3 (*)
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6 (*)
  - LL_APB1_GRP1_PERIPH_TIM7 (*)
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_WWDG

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR BKPEN LL_APB1_GRP1_EnableClock
- APB1ENR CAN1EN LL_APB1_GRP1_EnableClock
- APB1ENR CAN2EN LL_APB1_GRP1_EnableClock
- APB1ENR CECEN LL_APB1_GRP1_EnableClock
- APB1ENR DACEN LL_APB1_GRP1_EnableClock
- APB1ENR I2C1EN LL_APB1_GRP1_EnableClock
- APB1ENR I2C2EN LL_APB1_GRP1_EnableClock
- APB1ENR PWREN LL_APB1_GRP1_EnableClock
- APB1ENR SPI2EN LL_APB1_GRP1_EnableClock
- APB1ENR SPI3EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM12EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM13EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM14EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM2EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM3EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM4EN LL_APB1_GRP1_EnableClock

- APB1ENR TIM5EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM6EN LL_APB1_GRP1_EnableClock
- APB1ENR TIM7EN LL_APB1_GRP1_EnableClock
- APB1ENR UART4EN LL_APB1_GRP1_EnableClock
- APB1ENR UART5EN LL_APB1_GRP1_EnableClock
- APB1ENR USART2EN LL_APB1_GRP1_EnableClock
- APB1ENR USART3EN LL_APB1_GRP1_EnableClock
- APB1ENR USBEN LL_APB1_GRP1_EnableClock
- APB1ENR WWDGEN LL_APB1_GRP1_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name **__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)**

Function description Check if APB1 peripheral clock is enabled or not.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP1_PERIPH_BKP
  - LL_APB1_GRP1_PERIPH_CAN1 (*)
  - LL_APB1_GRP1_PERIPH_CAN2 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_DAC1 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_SPI3 (*)
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6 (*)
  - LL_APB1_GRP1_PERIPH_TIM7 (*)
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_WWDG

Return values
- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:
- APB1ENR BKPEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR CAN1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR CAN2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR CECEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR DACEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR PWREN LL_APB1_GRP1_IsEnabledClock

- APB1ENR SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR SPI3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM12EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM13EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM14EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR UART4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR UART5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR USBEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR WWDGEN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name **__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)**

Function description Disable APB1 peripherals clock.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP1_PERIPH_BKP
  - LL_APB1_GRP1_PERIPH_CAN1 (*)
  - LL_APB1_GRP1_PERIPH_CAN2 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_DAC1 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_SPI3 (*)
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6 (*)
  - LL_APB1_GRP1_PERIPH_TIM7 (*)
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_WWDG

Return values
- **None:**

Reference Manual to LL API cross reference:

- APB1ENR BKPEN LL_APB1_GRP1_DisableClock
- APB1ENR CAN1EN LL_APB1_GRP1_DisableClock
- APB1ENR CAN2EN LL_APB1_GRP1_DisableClock
- APB1ENR CECEN LL_APB1_GRP1_DisableClock
- APB1ENR DACEN LL_APB1_GRP1_DisableClock
- APB1ENR I2C1EN LL_APB1_GRP1_DisableClock
- APB1ENR I2C2EN LL_APB1_GRP1_DisableClock
- APB1ENR PWREN LL_APB1_GRP1_DisableClock
- APB1ENR SPI2EN LL_APB1_GRP1_DisableClock
- APB1ENR SPI3EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM12EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM13EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM14EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM2EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM3EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM4EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM5EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM6EN LL_APB1_GRP1_DisableClock
- APB1ENR TIM7EN LL_APB1_GRP1_DisableClock
- APB1ENR UART4EN LL_APB1_GRP1_DisableClock
- APB1ENR UART5EN LL_APB1_GRP1_DisableClock
- APB1ENR USART2EN LL_APB1_GRP1_DisableClock
- APB1ENR USART3EN LL_APB1_GRP1_DisableClock
- APB1ENR USBEN LL_APB1_GRP1_DisableClock
- APB1ENR WWDGEN LL_APB1_GRP1_DisableClock

LL_APB1_GRP1_ForceReset

Function name **__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)**

Function description Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_BKP
  - LL_APB1_GRP1_PERIPH_CAN1 (*)
  - LL_APB1_GRP1_PERIPH_CAN2 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_DAC1 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_SPI3 (*)
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6 (*)

  - LL_APB1_GRP1_PERIPH_TIM7 (*)
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_WWDG

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB1RSTR BKPRST LL_APB1_GRP1_ForceReset
- APB1RSTR CAN1RST LL_APB1_GRP1_ForceReset
- APB1RSTR CAN2RST LL_APB1_GRP1_ForceReset
- APB1RSTR CECRST LL_APB1_GRP1_ForceReset
- APB1RSTR DACRST LL_APB1_GRP1_ForceReset
- APB1RSTR I2C1RST LL_APB1_GRP1_ForceReset
- APB1RSTR I2C2RST LL_APB1_GRP1_ForceReset
- APB1RSTR PWRRST LL_APB1_GRP1_ForceReset
- APB1RSTR SPI2RST LL_APB1_GRP1_ForceReset
- APB1RSTR SPI3RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM12RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM13RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM14RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM2RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM3RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM4RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM5RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM6RST LL_APB1_GRP1_ForceReset
- APB1RSTR TIM7RST LL_APB1_GRP1_ForceReset
- APB1RSTR UART4RST LL_APB1_GRP1_ForceReset
- APB1RSTR UART5RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART2RST LL_APB1_GRP1_ForceReset
- APB1RSTR USART3RST LL_APB1_GRP1_ForceReset
- APB1RSTR USBRST LL_APB1_GRP1_ForceReset
- APB1RSTR WWDGRST LL_APB1_GRP1_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name
__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description
Release APB1 peripherals reset.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_BKP
  - LL_APB1_GRP1_PERIPH_CAN1 (*)
  - LL_APB1_GRP1_PERIPH_CAN2 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_DAC1 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_SPI2 (*)

- LL_APB1_GRP1_PERIPH_SPI3 (*)
- LL_APB1_GRP1_PERIPH_TIM12 (*)
- LL_APB1_GRP1_PERIPH_TIM13 (*)
- LL_APB1_GRP1_PERIPH_TIM14 (*)
- LL_APB1_GRP1_PERIPH_TIM2
- LL_APB1_GRP1_PERIPH_TIM3
- LL_APB1_GRP1_PERIPH_TIM4 (*)
- LL_APB1_GRP1_PERIPH_TIM5 (*)
- LL_APB1_GRP1_PERIPH_TIM6 (*)
- LL_APB1_GRP1_PERIPH_TIM7 (*)
- LL_APB1_GRP1_PERIPH_UART4 (*)
- LL_APB1_GRP1_PERIPH_UART5 (*)
- LL_APB1_GRP1_PERIPH_USART2
- LL_APB1_GRP1_PERIPH_USART3 (*)
- LL_APB1_GRP1_PERIPH_USB (*)
- LL_APB1_GRP1_PERIPH_WWDG

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR BKPRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR CAN1RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR CAN2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR CECRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR DACRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR I2C1RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR I2C2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR PWRRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR SPI2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR SPI3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM12RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM13RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM14RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM6RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR TIM7RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR UART4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR UART5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USART2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USART3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR USBRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR WWDGRST LL_APB1_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB2_GRP1_PERIPH_ADC1

  - LL_APB2_GRP1_PERIPH_ADC2 (*)
  - LL_APB2_GRP1_PERIPH_ADC3 (*)
  - LL_APB2_GRP1_PERIPH_AFIO
  - LL_APB2_GRP1_PERIPH_GPIOA
  - LL_APB2_GRP1_PERIPH_GPIOB
  - LL_APB2_GRP1_PERIPH_GPIOC
  - LL_APB2_GRP1_PERIPH_GPIOD
  - LL_APB2_GRP1_PERIPH_GPIOE (*)
  - LL_APB2_GRP1_PERIPH_GPIOF (*)
  - LL_APB2_GRP1_PERIPH_GPIOG (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM10 (*)
  - LL_APB2_GRP1_PERIPH_TIM11 (*)
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_TIM9 (*)
  - LL_APB2_GRP1_PERIPH_USART1

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB2ENR ADC1EN LL_APB2_GRP1_EnableClock
- APB2ENR ADC2EN LL_APB2_GRP1_EnableClock
- APB2ENR ADC3EN LL_APB2_GRP1_EnableClock
- APB2ENR AFIOEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPAEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPBEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPCEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPDEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPEEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPFEN LL_APB2_GRP1_EnableClock
- APB2ENR IOPGEN LL_APB2_GRP1_EnableClock
- APB2ENR SPI1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM10EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM11EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM9EN LL_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name
__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description
Check if APB2 peripheral clock is enabled or not.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB2_GRP1_PERIPH_ADC1

- LL_APB2_GRP1_PERIPH_ADC2 (*)
- LL_APB2_GRP1_PERIPH_ADC3 (*)
- LL_APB2_GRP1_PERIPH_AFIO
- LL_APB2_GRP1_PERIPH_GPIOA
- LL_APB2_GRP1_PERIPH_GPIOB
- LL_APB2_GRP1_PERIPH_GPIOC
- LL_APB2_GRP1_PERIPH_GPIOD
- LL_APB2_GRP1_PERIPH_GPIOE (*)
- LL_APB2_GRP1_PERIPH_GPIOF (*)
- LL_APB2_GRP1_PERIPH_GPIOG (*)
- LL_APB2_GRP1_PERIPH_SPI1
- LL_APB2_GRP1_PERIPH_TIM10 (*)
- LL_APB2_GRP1_PERIPH_TIM11 (*)
- LL_APB2_GRP1_PERIPH_TIM15 (*)
- LL_APB2_GRP1_PERIPH_TIM16 (*)
- LL_APB2_GRP1_PERIPH_TIM17 (*)
- LL_APB2_GRP1_PERIPH_TIM1
- LL_APB2_GRP1_PERIPH_TIM8 (*)
- LL_APB2_GRP1_PERIPH_TIM9 (*)
- LL_APB2_GRP1_PERIPH_USART1

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB2ENR ADC1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR ADC2EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR ADC3EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR AFIOEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPAEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPBEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPCEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPDEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPEEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPFEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR IOPGEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM10EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM11EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM9EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB2_GRP1_PERIPH_ADC1

  - LL_APB2_GRP1_PERIPH_ADC2 (*)
  - LL_APB2_GRP1_PERIPH_ADC3 (*)
  - LL_APB2_GRP1_PERIPH_AFIO
  - LL_APB2_GRP1_PERIPH_GPIOA
  - LL_APB2_GRP1_PERIPH_GPIOB
  - LL_APB2_GRP1_PERIPH_GPIOC
  - LL_APB2_GRP1_PERIPH_GPIOD
  - LL_APB2_GRP1_PERIPH_GPIOE (*)
  - LL_APB2_GRP1_PERIPH_GPIOF (*)
  - LL_APB2_GRP1_PERIPH_GPIOG (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM10 (*)
  - LL_APB2_GRP1_PERIPH_TIM11 (*)
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_TIM9 (*)
  - LL_APB2_GRP1_PERIPH_USART1

Return values
- **None:**

Reference Manual to LL API cross reference:
- APB2ENR ADC1EN LL_APB2_GRP1_DisableClock
- APB2ENR ADC2EN LL_APB2_GRP1_DisableClock
- APB2ENR ADC3EN LL_APB2_GRP1_DisableClock
- APB2ENR AFIOEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPAEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPBEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPCEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPDEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPEEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPFEN LL_APB2_GRP1_DisableClock
- APB2ENR IOPGEN LL_APB2_GRP1_DisableClock
- APB2ENR SPI1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM10EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM11EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM9EN LL_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name
__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description
Force APB2 peripherals reset.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_APB2_GRP1_PERIPH_ALL

  - LL_APB2_GRP1_PERIPH_ADC1
  - LL_APB2_GRP1_PERIPH_ADC2 (*)
  - LL_APB2_GRP1_PERIPH_ADC3 (*)
  - LL_APB2_GRP1_PERIPH_AFIO
  - LL_APB2_GRP1_PERIPH_GPIOA
  - LL_APB2_GRP1_PERIPH_GPIOB
  - LL_APB2_GRP1_PERIPH_GPIOC
  - LL_APB2_GRP1_PERIPH_GPIOD
  - LL_APB2_GRP1_PERIPH_GPIOE (*)
  - LL_APB2_GRP1_PERIPH_GPIOF (*)
  - LL_APB2_GRP1_PERIPH_GPIOG (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM10 (*)
  - LL_APB2_GRP1_PERIPH_TIM11 (*)
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_TIM9 (*)
  - LL_APB2_GRP1_PERIPH_USART1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR ADC1RST LL_APB2_GRP1_ForceReset
- APB2RSTR ADC2RST LL_APB2_GRP1_ForceReset
- APB2RSTR ADC3RST LL_APB2_GRP1_ForceReset
- APB2RSTR AFIORST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPARST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPBRST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPCRST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPDRST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPERST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPFRST LL_APB2_GRP1_ForceReset
- APB2RSTR IOPGRST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM10RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM11RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM9RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART1RST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.

  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_ADC1
  - LL_APB2_GRP1_PERIPH_ADC2 (*)
  - LL_APB2_GRP1_PERIPH_ADC3 (*)
  - LL_APB2_GRP1_PERIPH_AFIO
  - LL_APB2_GRP1_PERIPH_GPIOA
  - LL_APB2_GRP1_PERIPH_GPIOB
  - LL_APB2_GRP1_PERIPH_GPIOC
  - LL_APB2_GRP1_PERIPH_GPIOD
  - LL_APB2_GRP1_PERIPH_GPIOE (*)
  - LL_APB2_GRP1_PERIPH_GPIOF (*)
  - LL_APB2_GRP1_PERIPH_GPIOG (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM10 (*)
  - LL_APB2_GRP1_PERIPH_TIM11 (*)
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_TIM9 (*)
  - LL_APB2_GRP1_PERIPH_USART1

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR ADC1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR ADC2RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR ADC3RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR AFIORST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPARST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPBRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPCRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPDRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPERST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPFRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR IOPGRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM10RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM11RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM9RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USART1RST LL_APB2_GRP1_ReleaseReset

47.2 BUS Firmware driver defines

47.2.1 BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_CRC

LL_AHB1_GRP1_PERIPH_DMA1

LL_AHB1_GRP1_PERIPH_DMA2

LL_AHB1_GRP1_PERIPH_FLASH

LL_AHB1_GRP1_PERIPH_FSMC

LL_AHB1_GRP1_PERIPH_SDIO

LL_AHB1_GRP1_PERIPH_SRAM

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL

LL_APB1_GRP1_PERIPH_BKP

LL_APB1_GRP1_PERIPH_CAN1

LL_APB1_GRP1_PERIPH_DAC1

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_PWR

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_TIM12

LL_APB1_GRP1_PERIPH_TIM13

LL_APB1_GRP1_PERIPH_TIM14

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_USB

LL_APB1_GRP1_PERIPH_WWDG

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_ALL

LL_APB2_GRP1_PERIPH_ADC1

LL_APB2_GRP1_PERIPH_ADC2
LL_APB2_GRP1_PERIPH_ADC3
LL_APB2_GRP1_PERIPH_AFIO
LL_APB2_GRP1_PERIPH_GPIOA
LL_APB2_GRP1_PERIPH_GPIOB
LL_APB2_GRP1_PERIPH_GPIOC
LL_APB2_GRP1_PERIPH_GPIOD
LL_APB2_GRP1_PERIPH_GPIOE
LL_APB2_GRP1_PERIPH_GPIOF
LL_APB2_GRP1_PERIPH_GPIOG
LL_APB2_GRP1_PERIPH_SPI1
LL_APB2_GRP1_PERIPH_TIM10
LL_APB2_GRP1_PERIPH_TIM11
LL_APB2_GRP1_PERIPH_TIM1
LL_APB2_GRP1_PERIPH_TIM8
LL_APB2_GRP1_PERIPH_TIM9
LL_APB2_GRP1_PERIPH_USART1

48 LL CORTEX Generic Driver

48.1 CORTEX Firmware driver API description

48.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)**
Function description	This function checks if the Systick counter flag is active or not.
Return values	• **State:** of bit (1 or 0).
Notes	• It can be used in timeout function on application side.
Reference Manual to LL API cross reference:	• STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name	**__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)**
Function description	Configures the SysTick clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_SYSTICK_CLKSOURCE_HCLK_DIV8 – LL_SYSTICK_CLKSOURCE_HCLK
Return values	• **None:**
Reference Manual to LL API cross reference:	• STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)**
Function description	Get the SysTick clock source.
Return values	• **Returned:** value can be one of the following values: – LL_SYSTICK_CLKSOURCE_HCLK_DIV8 – LL_SYSTICK_CLKSOURCE_HCLK
Reference Manual to LL API cross reference:	• STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name	**__STATIC_INLINE void LL_SYSTICK_EnableIT (void)**
Function description	Enable SysTick exception request.

Return values	• **None:**
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name	**__STATIC_INLINE void LL_SYSTICK_DisableIT (void)**
Function description	Disable SysTick exception request.
Return values	• **None:**
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name	**__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)**
Function description	Checks if the SYSTICK interrupt is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name	**__STATIC_INLINE void LL_LPM_EnableSleep (void)**
Function description	Processor uses sleep as its low power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name	**__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)**
Function description	Processor uses deep sleep as its low power mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name	**__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)**
Function description	Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values	• **None:**
Notes	• Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name	**__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)**
Function description	Do not sleep when returning to Thread mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name	**__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)**
Function description	Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name	**__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)**
Function description	Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name	**__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)**
Function description	Enable a fault in System handler control register (SHCSR)
Parameters	• **Fault:** This parameter can be a combination of the following values: – LL_HANDLER_FAULT_USG – LL_HANDLER_FAULT_BUS – LL_HANDLER_FAULT_MEM

Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name	**__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)**
Function description	Disable a fault in System handler control register (SHCSR)
Parameters	• **Fault:** This parameter can be a combination of the following values: – LL_HANDLER_FAULT_USG – LL_HANDLER_FAULT_BUS – LL_HANDLER_FAULT_MEM
Return values	• **None:**
Reference Manual to LL API cross reference:	• SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)**
Function description	Get Implementer code.
Return values	• **Value:** should be equal to 0x41 for ARM
Reference Manual to LL API cross reference:	• SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)**
Function description	Get Variant number (The r value in the rnpn product revision identifier)
Return values	• **Value:** between 0 and 255 (0x1: revision 1, 0x2: revision 2)
Reference Manual to LL API cross reference:	• SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetConstant

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetConstant (void)**
Function description	Get Constant number.
Return values	• **Value:** should be equal to 0xF for Cortex-M3 devices
Reference Manual to LL API cross	• SCB_CPUID ARCHITECTURE LL_CPUID_GetConstant

reference:

LL_CPUID_GetParNo

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)**
Function description	Get Part number.
Return values	• **Value:** should be equal to 0xC23 for Cortex-M3
Reference Manual to LL API cross reference:	• SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name	**__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)**
Function description	Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)
Return values	• **Value:** between 0 and 255 (0x0: patch 0, 0x1: patch 1)
Reference Manual to LL API cross reference:	• SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name	**__STATIC_INLINE void LL_MPU_Enable (uint32_t Options)**
Function description	Enable MPU with input options.
Parameters	• **Options:** This parameter can be one of the following values: – LL_MPU_CTRL_HFNMI_PRIVDEF_NONE – LL_MPU_CTRL_HARDFAULT_NMI – LL_MPU_CTRL_PRIVILEGED_DEFAULT – LL_MPU_CTRL_HFNMI_PRIVDEF
Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name	**__STATIC_INLINE void LL_MPU_Disable (void)**
Function description	Disable MPU.
Return values	• **None:**
Reference Manual to LL API cross reference:	• MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)**

Function description: Check if MPU is enabled or not.

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_EnableRegion

Function name: **__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)**

Function description: Enable a MPU region.

Parameters:
- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values:
- **None:**

Reference Manual to LL API cross reference:
- MPU_RASR ENABLE LL_MPU_EnableRegion

LL_MPU_ConfigRegion

Function name: **__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t SubRegionDisable, uint32_t Address, uint32_t Attributes)**

Function description: Configure and enable a region.

Parameters:
- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Address:** Value of region base address
- **SubRegionDisable:** Sub-region disable value between Min_Data = 0x00 and Max_Data = 0xFF
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_REGION_SIZE_32B or LL_MPU_REGION_SIZE_64B or LL_MPU_REGION_SIZE_128B or LL_MPU_REGION_SIZE_256B or LL_MPU_REGION_SIZE_512B or

LL_MPU_REGION_SIZE_1KB or
LL_MPU_REGION_SIZE_2KB or
LL_MPU_REGION_SIZE_4KB or
LL_MPU_REGION_SIZE_8KB or
LL_MPU_REGION_SIZE_16KB or
LL_MPU_REGION_SIZE_32KB or
LL_MPU_REGION_SIZE_64KB or
LL_MPU_REGION_SIZE_128KB or
LL_MPU_REGION_SIZE_256KB or
LL_MPU_REGION_SIZE_512KB or
LL_MPU_REGION_SIZE_1MB or
LL_MPU_REGION_SIZE_2MB or
LL_MPU_REGION_SIZE_4MB or
LL_MPU_REGION_SIZE_8MB or
LL_MPU_REGION_SIZE_16MB or
LL_MPU_REGION_SIZE_32MB or
LL_MPU_REGION_SIZE_64MB or
LL_MPU_REGION_SIZE_128MB or
LL_MPU_REGION_SIZE_256MB or
LL_MPU_REGION_SIZE_512MB or
LL_MPU_REGION_SIZE_1GB or
LL_MPU_REGION_SIZE_2GB or
LL_MPU_REGION_SIZE_4GB
  - LL_MPU_REGION_NO_ACCESS or LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_PRIV_RW_URO or LL_MPU_REGION_FULL_ACCESS or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_PRIV_RO_URO
  - LL_MPU_TEX_LEVEL0 or LL_MPU_TEX_LEVEL1 or LL_MPU_TEX_LEVEL2 or LL_MPU_TEX_LEVEL4
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_SHAREABLE or LL_MPU_ACCESS_NOT_SHAREABLE
  - LL_MPU_ACCESS_CACHEABLE or LL_MPU_ACCESS_NOT_CACHEABLE
  - LL_MPU_ACCESS_BUFFERABLE or LL_MPU_ACCESS_NOT_BUFFERABLE

Return values
- **None:**

Reference Manual to LL API cross reference:
- MPU_RNR REGION LL_MPU_ConfigRegion
- MPU_RBAR REGION LL_MPU_ConfigRegion
- MPU_RBAR ADDR LL_MPU_ConfigRegion
- MPU_RASR XN LL_MPU_ConfigRegion
- MPU_RASR AP LL_MPU_ConfigRegion
- MPU_RASR S LL_MPU_ConfigRegion
- MPU_RASR C LL_MPU_ConfigRegion
- MPU_RASR B LL_MPU_ConfigRegion
- MPU_RASR SIZE LL_MPU_ConfigRegion

LL_MPU_DisableRegion

Function name **__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t**

Region)

Function description: Disable a region.

Parameters:
- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values:
- **None:**

Reference Manual to LL API cross reference:
- MPU_RNR REGION LL_MPU_DisableRegion
- MPU_RASR ENABLE LL_MPU_DisableRegion

48.2 CORTEX Firmware driver defines

48.2.1 CORTEX

MPU Bufferable Access

LL_MPU_ACCESS_BUFFERABLE	Bufferable memory attribute
LL_MPU_ACCESS_NOT_BUFFERABLE	Not Bufferable memory attribute

MPU Cacheable Access

LL_MPU_ACCESS_CACHEABLE	Cacheable memory attribute
LL_MPU_ACCESS_NOT_CACHEABLE	Not Cacheable memory attribute

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8	AHB clock divided by 8 selected as SysTick clock source.
LL_SYSTICK_CLKSOURCE_HCLK	AHB clock selected as SysTick clock source.

MPU Control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE	Disable NMI and privileged SW access
LL_MPU_CTRL_HARDFAULT_NMI	Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers
LL_MPU_CTRL_PRIVILEGED_DEFAULT	Enable privileged software access to default memory map
LL_MPU_CTRL_HFNMI_PRIVDEF	Enable NMI and privileged SW access

Handler Fault type

LL_HANDLER_FAULT_USG	Usage fault
LL_HANDLER_FAULT_BUS	Bus fault
LL_HANDLER_FAULT_MEM	Memory management fault

MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE	Instruction fetches enabled
LL_MPU_INSTRUCTION_ACCESS_DISABLE	Instruction fetches disabled

MPU Region Number

LL_MPU_REGION_NUMBER0	REGION Number 0
LL_MPU_REGION_NUMBER1	REGION Number 1
LL_MPU_REGION_NUMBER2	REGION Number 2
LL_MPU_REGION_NUMBER3	REGION Number 3
LL_MPU_REGION_NUMBER4	REGION Number 4
LL_MPU_REGION_NUMBER5	REGION Number 5
LL_MPU_REGION_NUMBER6	REGION Number 6
LL_MPU_REGION_NUMBER7	REGION Number 7

MPU Region Privileges

LL_MPU_REGION_NO_ACCESS	No access
LL_MPU_REGION_PRIV_RW	RW privileged (privileged access only)
LL_MPU_REGION_PRIV_RW_URO	RW privileged - RO user (Write in a user program generates a fault)
LL_MPU_REGION_FULL_ACCESS	RW privileged & user (Full access)
LL_MPU_REGION_PRIV_RO	RO privileged (privileged read only)
LL_MPU_REGION_PRIV_RO_URO	RO privileged & user (read only)

MPU Region Size

LL_MPU_REGION_SIZE_32B	32B Size of the MPU protection region
LL_MPU_REGION_SIZE_64B	64B Size of the MPU protection region
LL_MPU_REGION_SIZE_128B	128B Size of the MPU protection region
LL_MPU_REGION_SIZE_256B	256B Size of the MPU protection region
LL_MPU_REGION_SIZE_512B	512B Size of the MPU protection region
LL_MPU_REGION_SIZE_1KB	1KB Size of the MPU protection region
LL_MPU_REGION_SIZE_2KB	2KB Size of the MPU protection region
LL_MPU_REGION_SIZE_4KB	4KB Size of the MPU protection region
LL_MPU_REGION_SIZE_8KB	8KB Size of the MPU protection region
LL_MPU_REGION_SIZE_16KB	16KB Size of the MPU protection region
LL_MPU_REGION_SIZE_32KB	32KB Size of the MPU protection region
LL_MPU_REGION_SIZE_64KB	64KB Size of the MPU protection region
LL_MPU_REGION_SIZE_128KB	128KB Size of the MPU protection region
LL_MPU_REGION_SIZE_256KB	256KB Size of the MPU protection region
LL_MPU_REGION_SIZE_512KB	512KB Size of the MPU protection region
LL_MPU_REGION_SIZE_1MB	1MB Size of the MPU protection region
LL_MPU_REGION_SIZE_2MB	2MB Size of the MPU protection region

LL_MPU_REGION_SIZE_4MB	4MB Size of the MPU protection region
LL_MPU_REGION_SIZE_8MB	8MB Size of the MPU protection region
LL_MPU_REGION_SIZE_16MB	16MB Size of the MPU protection region
LL_MPU_REGION_SIZE_32MB	32MB Size of the MPU protection region
LL_MPU_REGION_SIZE_64MB	64MB Size of the MPU protection region
LL_MPU_REGION_SIZE_128MB	128MB Size of the MPU protection region
LL_MPU_REGION_SIZE_256MB	256MB Size of the MPU protection region
LL_MPU_REGION_SIZE_512MB	512MB Size of the MPU protection region
LL_MPU_REGION_SIZE_1GB	1GB Size of the MPU protection region
LL_MPU_REGION_SIZE_2GB	2GB Size of the MPU protection region
LL_MPU_REGION_SIZE_4GB	4GB Size of the MPU protection region

MPU Shareable Access

LL_MPU_ACCESS_SHAREABLE	Shareable memory attribute
LL_MPU_ACCESS_NOT_SHAREABLE	Not Shareable memory attribute

MPU TEX Level

LL_MPU_TEX_LEVEL0	b000 for TEX bits
LL_MPU_TEX_LEVEL1	b001 for TEX bits
LL_MPU_TEX_LEVEL2	b010 for TEX bits
LL_MPU_TEX_LEVEL4	b100 for TEX bits

49 LL CRC Generic Driver

49.1 CRC Firmware driver API description

49.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name	**__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)**
Function description	Reset the CRC calculation unit.
Parameters	• **CRCx:** CRC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_FeedData32

Function name	**__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)**
Function description	Write given 32-bit data to the CRC calculator.
Parameters	• **CRCx:** CRC Instance • **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_CRC_FeedData32

LL_CRC_ReadData32

Function name	**__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)**
Function description	Return current CRC calculation result.
Parameters	• **CRCx:** CRC Instance
Return values	• **Current:** CRC calculation result as stored in CRC_DR register (32 bits).
Reference Manual to LL API cross reference:	• DR DR LL_CRC_ReadData32

LL_CRC_Read_IDR

Function name	**__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef**

*** CRCx)**

Function description	Return data stored in the Independent Data(IDR) register.
Parameters	• **CRCx:** CRC Instance
Return values	• **Value:** stored in CRC_IDR register (General-purpose 8-bit data register).
Notes	• This register can be used as a temporary storage location for one byte.
Reference Manual to LL API cross reference:	• IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name	**__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)**
Function description	Store data in the Independent Data(IDR) register.
Parameters	• **CRCx:** CRC Instance • **InData:** value to be stored in CRC_IDR register (8-bit) between between Min_Data=0 and Max_Data=0xFF
Return values	• **None:**
Notes	• This register can be used as a temporary storage location for one byte.
Reference Manual to LL API cross reference:	• IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name	**ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)**
Function description	De-initialize CRC registers (Registers restored to their default values).
Parameters	• **CRCx:** CRC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: CRC registers are de-initialized – ERROR: CRC registers are not de-initialized

49.2 CRC Firmware driver defines

49.2.1 CRC

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

50 LL DAC Generic Driver

50.1 DAC Firmware driver registers structures

50.1.1 LL_DAC_InitTypeDef

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t WaveAutoGeneration***
- ***uint32_t WaveAutoGenerationConfig***
- ***uint32_t OutputBuffer***

Field Documentation

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource***
  Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function **LL_DAC_SetTriggerSource()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration***
  Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_WAVE_AUTO_GENERATION_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetWaveAutoGeneration()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig***
  Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of ***DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS*** If waveform automatic generation mode is set to triangle, this parameter can be a value of ***DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE***
  Note: If waveform automatic generation mode is disabled, this parameter is discarded. This feature can be modified afterwards using unitary function **LL_DAC_SetWaveNoiseLFSR()** or **LL_DAC_SetWaveTriangleAmplitude()**, depending on the wave automatic generation selected.
- ***uint32_t LL_DAC_InitTypeDef::OutputBuffer***
  Set the output buffer for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_BUFFER***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputBuffer()**.

50.2 DAC Firmware driver API description

50.2.1 Detailed description of functions

LL_DAC_SetTriggerSource

Function name	**__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)**
Function description	Set the conversion trigger source for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following

values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriggerSource:** This parameter can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM5_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE9

Return values
- **None:**

Notes
- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR TSEL1 LL_DAC_SetTriggerSource
- CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name
__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Get the conversion trigger source for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM5_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE9

Notes
- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- Availability of parameters of trigger sources from timer

depends on timers availability on the selected device.

Reference Manual to LL API cross reference:
- CR TSEL1 LL_DAC_GetTriggerSource
- CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name **__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)**

Function description Set the waveform automatic generation mode for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **WaveAutoGeneration:** This parameter can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR WAVE1 LL_DAC_SetWaveAutoGeneration
- CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name **__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel)**

Function description Get the waveform automatic generation mode for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **Returned:** value can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Reference Manual to LL API cross reference:
- CR WAVE1 LL_DAC_GetWaveAutoGeneration
- CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name **__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR**

(DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)

Function description: Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters:

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **NoiseLFSRMask:** This parameter can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Return values:

- **None:**

Notes:

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveNoiseLFSR
- CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name: **__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel)**

Function description: Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters:

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values:

- **Returned:** value can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0

  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name

__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)

Function description

Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name
__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **Returned:** value can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Reference Manual to LL API cross reference:
- CR MAMP1 LL_DAC_GetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_SetOutputBuffer

Function name
__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)

Function description
Set the output buffer for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR BOFF1 LL_DAC_SetOutputBuffer
- CR BOFF2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name	**__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Get the output buffer state for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2
Return values	• **Returned:** value can be one of the following values: – LL_DAC_OUTPUT_BUFFER_ENABLE – LL_DAC_OUTPUT_BUFFER_DISABLE
Reference Manual to LL API cross reference:	• CR BOFF1 LL_DAC_GetOutputBuffer • CR BOFF2 LL_DAC_GetOutputBuffer

LL_DAC_EnableDMAReq

Function name	**__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Enable DAC DMA transfer request of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2
Return values	• **None:**
Notes	• To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().
Reference Manual to LL API cross reference:	• CR DMAEN1 LL_DAC_EnableDMAReq • CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name	**__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Disable DAC DMA transfer request of the selected channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2
Return values	• **None:**
Notes	• To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:
- CR DMAEN1 LL_DAC_DisableDMAReq
- CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name: **__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)**

Function description: Get DAC DMA transfer request state of the selected channel.

Parameters:
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CR DMAEN1 LL_DAC_IsDMAReqEnabled
- CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_DMA_GetRegAddr

Function name: **__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)**

Function description: Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.

Parameters:
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Register:** This parameter can be one of the following values:
  - LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  - LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  - LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values:
- **DAC:** register address

Notes:
- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example:
  LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:
- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr

- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name
__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Enable DAC selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **None:**

Notes
- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:
- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name
__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Disable DAC selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR EN1 LL_DAC_Disable
- CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name
__STATIC_INLINE uint32_t LL_DAC_IsEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Get DAC enable state of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CR EN1 LL_DAC_IsEnabled
- CR EN2 LL_DAC_IsEnabled

LL_DAC_EnableTrigger

Function name
__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Enable DAC trigger of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **None:**

Notes
- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross reference:
- CR TEN1 LL_DAC_EnableTrigger
- CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name
__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Disable DAC trigger of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR TEN1 LL_DAC_DisableTrigger
- CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name
__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Get DAC trigger state of the selected channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CR TEN1 LL_DAC_IsTriggerEnabled
- CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name
__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description
Trig DAC conversion by software for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can a combination of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values
- **None:**

Notes
- Preliminarily, DAC trigger must be set to software trigger using function LL_DAC_SetTriggerSource() with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)

Reference Manual to LL API cross reference:
- SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion
- SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion

LL_DAC_ConvertData12RightAligned

Function name
__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description
Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values
- **None:**

Reference Manual to LL API cross reference:
- DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned
- DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name: **__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**

Function description: Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters:
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values:
- **None:**

Reference Manual to LL API cross reference:
- DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned
- DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name: **__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)**

Function description: Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters:
- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values:
- **None:**

Reference Manual to LL API cross reference:
- DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned
- DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name: **__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**

Function description: Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF • **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned • DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**
Function description	Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF • **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned • DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name	**__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)**
Function description	Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.
Parameters	• **DACx:** DAC instance • **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF • **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned • DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name	**__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)**
Function description	Retrieve output data currently generated for the selected DAC channel.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2
Return values	• **Value:** between Min_Data=0x000 and Max_Data=0xFFF
Notes	• Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).
Reference Manual to LL API cross reference:	• DOR1 DACC1DOR LL_DAC_RetrieveOutputData • DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_DeInit

Function name	**ErrorStatus LL_DAC_DeInit (DAC_TypeDef * DACx)**
Function description	De-initialize registers of the selected DAC instance to their default reset values.
Parameters	• **DACx:** DAC instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DAC registers are de-initialized – ERROR: not applicable

LL_DAC_Init

Function name	**ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, LL_DAC_InitTypeDef * DAC_InitStruct)**
Function description	Initialize some features of DAC instance.
Parameters	• **DACx:** DAC instance • **DAC_Channel:** This parameter can be one of the following values: – LL_DAC_CHANNEL_1 – LL_DAC_CHANNEL_2 • **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: DAC registers are initialized – ERROR: DAC registers are not initialized
Notes	• The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC instance must be disabled.

LL_DAC_StructInit

Function name **void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)**

Function description Set each LL_DAC_InitTypeDef field to default value.

Parameters
- **DAC_InitStruct:** pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.

Return values
- **None:**

50.3 DAC Firmware driver defines

50.3.1 DAC

DAC channels

LL_DAC_CHANNEL_1	DAC channel 1
LL_DAC_CHANNEL_2	DAC channel 2

Definitions of DAC hardware constraints delays

LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US	Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)
LL_DAC_DELAY_VOLTAGE_SETTLING_US	Delay for DAC channel voltage settling time

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE	The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption
LL_DAC_OUTPUT_BUFFER_DISABLE	The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED	DAC channel data holding register 12 bits right aligned
LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED	DAC channel data holding register 12 bits left aligned
LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED	DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B	DAC channel resolution 12 bits
LL_DAC_RESOLUTION_8B	DAC channel resolution 8 bits

DAC trigger source

LL_DAC_TRIG_SOFTWARE	DAC channel conversion trigger internal (SW start)
LL_DAC_TRIG_EXT_TIM3_TRGO	DAC channel conversion trigger from external IP: TIM3 TRGO.

LL_DAC_TRIG_EXT_TIM15_TRGO	DAC channel conversion trigger from external IP: TIM15 TRGO.
LL_DAC_TRIG_EXT_TIM2_TRGO	DAC channel conversion trigger from external IP: TIM2 TRGO.
LL_DAC_TRIG_EXT_TIM8_TRGO	DAC channel conversion trigger from external IP: TIM8 TRGO.
LL_DAC_TRIG_EXT_TIM4_TRGO	DAC channel conversion trigger from external IP: TIM4 TRGO.
LL_DAC_TRIG_EXT_TIM6_TRGO	DAC channel conversion trigger from external IP: TIM6 TRGO.
LL_DAC_TRIG_EXT_TIM7_TRGO	DAC channel conversion trigger from external IP: TIM7 TRGO.
LL_DAC_TRIG_EXT_TIM5_TRGO	DAC channel conversion trigger from external IP: TIM5 TRGO.
LL_DAC_TRIG_EXT_EXTI_LINE9	DAC channel conversion trigger from external IP: external interrupt line 9.

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE	DAC channel wave auto generation mode disabled.
LL_DAC_WAVE_AUTO_GENERATION_NOISE	DAC channel wave auto generation mode enabled, set generated noise waveform.
LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE	DAC channel wave auto generation mode enabled, set generated triangle waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0	Noise wave generation, unmask LFSR bit0, for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS1_0	Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS2_0	Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS3_0	Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS4_0	Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS5_0	Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS6_0	Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS7_0	Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS8_0	Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS9_0	Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS10_0	Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel
LL_DAC_NOISE_LFSR_UNMASK_BITS11_0	Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1	Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_3	Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_7	Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_15	Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_31	Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_63	Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_127	Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_255	Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_511	Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_1023	Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_2047	Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel
LL_DAC_TRIANGLE_AMPLITUDE_4095	Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB	**Description:** • Helper macro to get DAC channel number

in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return value:

- 1...2

Notes:

- The input can be a value from functions where a channel number is returned.

__LL_DAC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: 1...2

Return value:

- Returned: value can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

__LL_DAC_DIGITAL_SCALE

Description:

- Helper macro to define the DAC conversion data full-scale digital value corresponding to the selected DAC resolution.

Parameters:

- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- DAC conversion data full-scale corresponds to voltage range determined

by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

- Helper macro to calculate the DAC conversion data (unit: digital value) corresponding to a voltage (unit: mVolt).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

51 LL DMA Generic Driver

51.1 DMA Firmware driver registers structures

51.1.1 LL_DMA_InitTypeDef

Data Fields

- ***uint32_t PeriphOrM2MSrcAddress***
- ***uint32_t MemoryOrM2MDstAddress***
- ***uint32_t Direction***
- ***uint32_t Mode***
- ***uint32_t PeriphOrM2MSrcIncMode***
- ***uint32_t MemoryOrM2MDstIncMode***
- ***uint32_t PeriphOrM2MSrcDataSize***
- ***uint32_t MemoryOrM2MDstDataSize***
- ***uint32_t NbData***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcAddress***
  Specifies the peripheral base address for DMA transfer or as Source base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstAddress***
  Specifies the memory base address for DMA transfer or as Destination base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_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_LL_EC_DIRECTION***This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::Mode***
  Specifies the normal or circular operation mode. This parameter can be a value of ***DMA_LL_EC_MODE***
  Note:: The circular buffer mode cannot be used if the memory to memory data transfer direction is configured on the selected Channel This feature can be modified afterwards using unitary function **LL_DMA_SetMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcIncMode***
  Specifies whether the Peripheral address or Source address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_PERIPH***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstIncMode***
  Specifies whether the Memory address or Destination address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_MEMORY***This feature can be modified afterwards using unitary function **LL_DMA_SetMemoryIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcDataSize***
  Specifies the Peripheral data size alignment or Source data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a

value of ***DMA_LL_EC_PDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphSize()**.

- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstDataSize***
  Specifies the Memory data size alignment or Destination data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_MDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetMemorySize()**.
- ***uint32_t LL_DMA_InitTypeDef::NbData***
  Specifies the number of data to transfer, in data unit. The data unit is equal to the source buffer configuration set in PeripheralSize or MemorySize parameters depending in the transfer direction. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFFThis feature can be modified afterwards using unitary function **LL_DMA_SetDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***
  Specifies the channel priority level. This parameter can be a value of ***DMA_LL_EC_PRIORITY***This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.

51.2 DMA Firmware driver API description

51.2.1 Detailed description of functions

LL_DMA_EnableChannel

Function name	**__STATIC_INLINE void LL_DMA_EnableChannel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable DMA channel.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name	**__STATIC_INLINE void LL_DMA_DisableChannel (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Disable DMA channel.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3

  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name
__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Check if DMA channel is enabled or disabled.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ConfigTransfer

Function name
__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description
Configure all parameters link to DMA transfer.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY
  - LL_DMA_MODE_NORMAL or

LL_DMA_MODE_CIRCULAR
- LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT
- LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT
- LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or LL_DMA_PDATAALIGN_WORD
- LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or LL_DMA_MDATAALIGN_WORD
- LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR DIR LL_DMA_ConfigTransfer
- CCR MEM2MEM LL_DMA_ConfigTransfer
- CCR CIRC LL_DMA_ConfigTransfer
- CCR PINC LL_DMA_ConfigTransfer
- CCR MINC LL_DMA_ConfigTransfer
- CCR PSIZE LL_DMA_ConfigTransfer
- CCR MSIZE LL_DMA_ConfigTransfer
- CCR PL LL_DMA_ConfigTransfer

LL_DMA_SetDataTransferDirection

Function name
__STATIC_INLINE void LL_DMA_SetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description
Set Data transfer direction (read from peripheral or from memory).

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR DIR LL_DMA_SetDataTransferDirection
- CCR MEM2MEM LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name
__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get Data transfer direction (read from peripheral or from memory).

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Reference Manual to LL API cross reference:
- CCR DIR LL_DMA_GetDataTransferDirection
- CCR MEM2MEM LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name
__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)

Function description
Set DMA mode circular or normal.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Mode:** This parameter can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR

Return values
- **None:**

Notes
- The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.

Reference Manual to LL API cross reference:
- CCR CIRC LL_DMA_SetMode

LL_DMA_GetMode

Function name
__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get DMA mode circular or normal.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Returned:** value can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR

Reference Manual to LL API cross reference:
- CCR CIRC LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name
__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)

Function description
Set Peripheral increment mode.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values:
  - LL_DMA_PERIPH_INCREMENT
  - LL_DMA_PERIPH_NOINCREMENT

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR PINC LL_DMA_SetPeriphIncMode

LL_DMA_GetPeriphIncMode

Function name
__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel)

Function description: Get Peripheral increment mode.

Parameters:
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values:
- **Returned:** value can be one of the following values:
  - LL_DMA_PERIPH_INCREMENT
  - LL_DMA_PERIPH_NOINCREMENT

Reference Manual to LL API cross reference:
- CCR PINC LL_DMA_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name: **__STATIC_INLINE void LL_DMA_SetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)**

Function description: Set Memory increment mode.

Parameters:
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **MemoryOrM2MDstIncMode:** This parameter can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Return values:
- **None:**

Reference Manual to LL API cross reference:
- CCR MINC LL_DMA_SetMemoryIncMode

LL_DMA_GetMemoryIncMode

Function name: **__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description: Get Memory increment mode.

Parameters:
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1

  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Returned:** value can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Reference Manual to LL API cross reference:
- CCR MINC LL_DMA_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name
__STATIC_INLINE void LL_DMA_SetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)

Function description
Set Peripheral size.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **PeriphOrM2MSrcDataSize:** This parameter can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name
__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get Peripheral size.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4

– LL_DMA_CHANNEL_5
– LL_DMA_CHANNEL_6
– LL_DMA_CHANNEL_7

Return values
- **Returned:** value can be one of the following values:
  – LL_DMA_PDATAALIGN_BYTE
  – LL_DMA_PDATAALIGN_HALFWORD
  – LL_DMA_PDATAALIGN_WORD

Reference Manual to LL API cross reference:
- CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name
__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)

Function description
Set Memory size.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
- **MemoryOrM2MDstDataSize:** This parameter can be one of the following values:
  – LL_DMA_MDATAALIGN_BYTE
  – LL_DMA_MDATAALIGN_HALFWORD
  – LL_DMA_MDATAALIGN_WORD

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name
__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get Memory size.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6

– LL_DMA_CHANNEL_7

Return values
- **Returned:** value can be one of the following values:
  – LL_DMA_MDATAALIGN_BYTE
  – LL_DMA_MDATAALIGN_HALFWORD
  – LL_DMA_MDATAALIGN_WORD

Reference Manual to LL API cross reference:
- CCR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetChannelPriorityLevel

Function name **__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)**

Function description Set Channel priority level.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
- **Priority:** This parameter can be one of the following values:
  – LL_DMA_PRIORITY_LOW
  – LL_DMA_PRIORITY_MEDIUM
  – LL_DMA_PRIORITY_HIGH
  – LL_DMA_PRIORITY_VERYHIGH

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR PL LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name **__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description Get Channel priority level.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7

Return values
- **Returned:** value can be one of the following values:
  – LL_DMA_PRIORITY_LOW

  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Reference Manual to LL API cross reference:
- CCR PL LL_DMA_GetChannelPriorityLevel

LL_DMA_SetDataLength

Function name
__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t NbData)

Function description
Set Number of data to transfer.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF

Return values
- **None:**

Notes
- This action has no effect if channel is enabled.

Reference Manual to LL API cross reference:
- CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name
__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get Number of data to transfer.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes
- Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.

Reference Manual to LL API cross
- CNDTR NDT LL_DMA_GetDataLength

reference:

LL_DMA_ConfigAddresses

Function name
__STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)

Function description
Configure the Source and Destination addresses.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values
- **None:**

Notes
- This API must not be called when the DMA channel is enabled.
- Each IP using DMA provides an API to get directly the register adress (LL_PPP_DMA_GetRegAddr).

Reference Manual to LL API cross reference:
- CPAR PA LL_DMA_ConfigAddresses
- CMAR MA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name
__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description
Set the Memory address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values
- **None:**

Notes
- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:
- CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name
__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)

Function description
Set the Peripheral address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **PeriphAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values
- **None:**

Notes
- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:
- CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name
__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get Memory address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3

- LL_DMA_CHANNEL_4
- LL_DMA_CHANNEL_5
- LL_DMA_CHANNEL_6
- LL_DMA_CHANNEL_7

Return values
- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes
- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:
- CMAR MA LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name
__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get Peripheral address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes
- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:
- CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name
__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description
Set the Memory to Memory Source address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6

  - LL_DMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values
- **None:**

Notes
- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:
- CPAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name
__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description
Set the Memory to Memory Destination address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values
- **None:**

Notes
- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:
- CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name
__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get the Memory to Memory Source address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3

  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes
- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:
- CPAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name
__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Get the Memory to Memory Destination address.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes
- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:
- CMAR MA LL_DMA_GetM2MDstAddress

LL_DMA_IsActiveFlag_GI1

Function name
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI1 (DMA_TypeDef * DMAx)

Function description
Get Channel 1 global interrupt flag.

Parameters
- **DMAx:** DMAx Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- ISR GIF1 LL_DMA_IsActiveFlag_GI1

LL_DMA_IsActiveFlag_GI2

Function name
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI2 (DMA_TypeDef * DMAx)

Function description	Get Channel 2 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF2 LL_DMA_IsActiveFlag_GI2

LL_DMA_IsActiveFlag_GI3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF3 LL_DMA_IsActiveFlag_GI3

LL_DMA_IsActiveFlag_GI4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF4 LL_DMA_IsActiveFlag_GI4

LL_DMA_IsActiveFlag_GI5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF5 LL_DMA_IsActiveFlag_GI5

LL_DMA_IsActiveFlag_GI6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 global interrupt flag.

Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF6 LL_DMA_IsActiveFlag_GI6

LL_DMA_IsActiveFlag_GI7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR GIF7 LL_DMA_IsActiveFlag_GI7

LL_DMA_IsActiveFlag_TC1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF5 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• ISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_HT1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• ISR HTIF4 LL_DMA_IsActiveFlag_HT4

reference:

LL_DMA_IsActiveFlag_HT5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF5 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_TE1

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)**
Function description	Get Channel 1 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF1 LL_DMA_IsActiveFlag_TE1

LL_DMA_IsActiveFlag_TE2

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (DMA_TypeDef * DMAx)**
Function description	Get Channel 2 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF2 LL_DMA_IsActiveFlag_TE2

LL_DMA_IsActiveFlag_TE3

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (DMA_TypeDef * DMAx)**
Function description	Get Channel 3 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF3 LL_DMA_IsActiveFlag_TE3

LL_DMA_IsActiveFlag_TE4

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (DMA_TypeDef * DMAx)**
Function description	Get Channel 4 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF4 LL_DMA_IsActiveFlag_TE4

LL_DMA_IsActiveFlag_TE5

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (DMA_TypeDef * DMAx)**
Function description	Get Channel 5 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF5 LL_DMA_IsActiveFlag_TE5

LL_DMA_IsActiveFlag_TE6

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)**
Function description	Get Channel 6 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF6 LL_DMA_IsActiveFlag_TE6

LL_DMA_IsActiveFlag_TE7

Function name	**__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (DMA_TypeDef * DMAx)**
Function description	Get Channel 7 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• ISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_ClearFlag_GI1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF1 LL_DMA_ClearFlag_GI1

LL_DMA_ClearFlag_GI2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF2 LL_DMA_ClearFlag_GI2

LL_DMA_ClearFlag_GI3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF3 LL_DMA_ClearFlag_GI3

LL_DMA_ClearFlag_GI4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF4 LL_DMA_ClearFlag_GI4

LL_DMA_ClearFlag_GI5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF5 LL_DMA_ClearFlag_GI5

LL_DMA_ClearFlag_GI6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF6 LL_DMA_ClearFlag_GI6

LL_DMA_ClearFlag_GI7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 global interrupt flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_TC1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 transfer complete flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_HT1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 half transfer flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_TE1

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 1 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 2 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 3 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 4 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 5 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 6 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name	**__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)**
Function description	Clear Channel 7 transfer error flag.
Parameters	• **DMAx:** DMAx Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• IFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_EnableIT_TC

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Transfer complete interrupt.
Parameters	• **DMAx:** DMAx Instance • **Channel:** This parameter can be one of the following values: – LL_DMA_CHANNEL_1 – LL_DMA_CHANNEL_2 – LL_DMA_CHANNEL_3 – LL_DMA_CHANNEL_4 – LL_DMA_CHANNEL_5 – LL_DMA_CHANNEL_6 – LL_DMA_CHANNEL_7
Return values	• **None:**
Reference Manual to LL API cross reference:	• CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_EnableIT_HT

Function name	**__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**
Function description	Enable Half transfer interrupt.
Parameters	• **DMAx:** DMAx Instance

- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TE

Function name
__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Enable Transfer error interrupt.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR TEIE LL_DMA_EnableIT_TE

LL_DMA_DisableIT_TC

Function name
__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Disable Transfer complete interrupt.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **None:**

Reference Manual to
- CCR TCIE LL_DMA_DisableIT_TC

LL API cross reference:

LL_DMA_DisableIT_HT

Function name **__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description Disable Half transfer interrupt.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TE

Function name **__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description Disable Transfer error interrupt.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR TEIE LL_DMA_DisableIT_TE

LL_DMA_IsEnabledIT_TC

Function name **__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description Check if Transfer complete Interrupt is enabled.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:

  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_IsEnabledIT_HT

Function name
__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Check if Half transfer Interrupt is enabled.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TE

Function name
__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)

Function description
Check if Transfer error Interrupt is enabled.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross
- CCR TEIE LL_DMA_IsEnabledIT_TE

reference:

LL_DMA_Init

Function name **uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)**

Function description Initialize the DMA registers according to the specified parameters in DMA_InitStruct.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are initialized
  - ERROR: Not applicable

Notes
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name **uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)**

Function description De-initialize the DMA registers to their default reset values.

Parameters
- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are de-initialized
  - ERROR: DMA registers are not de-initialized

LL_DMA_StructInit

Function name **void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)**

Function description Set each LL_DMA_InitTypeDef field to default value.

Parameters
- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values
- **None:**

51.3 DMA Firmware driver defines

51.3.1 DMA

CHANNEL

LL_DMA_CHANNEL_1	DMA Channel 1
LL_DMA_CHANNEL_2	DMA Channel 2
LL_DMA_CHANNEL_3	DMA Channel 3
LL_DMA_CHANNEL_4	DMA Channel 4
LL_DMA_CHANNEL_5	DMA Channel 5
LL_DMA_CHANNEL_6	DMA Channel 6
LL_DMA_CHANNEL_7	DMA Channel 7
LL_DMA_CHANNEL_ALL	DMA Channel all (used only for function

Clear Flags Defines

LL_DMA_IFCR_CGIF1	Channel 1 global flag
LL_DMA_IFCR_CTCIF1	Channel 1 transfer complete flag
LL_DMA_IFCR_CHTIF1	Channel 1 half transfer flag
LL_DMA_IFCR_CTEIF1	Channel 1 transfer error flag
LL_DMA_IFCR_CGIF2	Channel 2 global flag
LL_DMA_IFCR_CTCIF2	Channel 2 transfer complete flag
LL_DMA_IFCR_CHTIF2	Channel 2 half transfer flag
LL_DMA_IFCR_CTEIF2	Channel 2 transfer error flag
LL_DMA_IFCR_CGIF3	Channel 3 global flag
LL_DMA_IFCR_CTCIF3	Channel 3 transfer complete flag
LL_DMA_IFCR_CHTIF3	Channel 3 half transfer flag
LL_DMA_IFCR_CTEIF3	Channel 3 transfer error flag
LL_DMA_IFCR_CGIF4	Channel 4 global flag
LL_DMA_IFCR_CTCIF4	Channel 4 transfer complete flag
LL_DMA_IFCR_CHTIF4	Channel 4 half transfer flag
LL_DMA_IFCR_CTEIF4	Channel 4 transfer error flag
LL_DMA_IFCR_CGIF5	Channel 5 global flag
LL_DMA_IFCR_CTCIF5	Channel 5 transfer complete flag
LL_DMA_IFCR_CHTIF5	Channel 5 half transfer flag
LL_DMA_IFCR_CTEIF5	Channel 5 transfer error flag

LL_DMA_IFCR_CGIF6	Channel 6 global flag
LL_DMA_IFCR_CTCIF6	Channel 6 transfer complete flag
LL_DMA_IFCR_CHTIF6	Channel 6 half transfer flag
LL_DMA_IFCR_CTEIF6	Channel 6 transfer error flag
LL_DMA_IFCR_CGIF7	Channel 7 global flag
LL_DMA_IFCR_CTCIF7	Channel 7 transfer complete flag
LL_DMA_IFCR_CHTIF7	Channel 7 half transfer flag
LL_DMA_IFCR_CTEIF7	Channel 7 transfer error flag

Transfer Direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY	Peripheral to memory direction
LL_DMA_DIRECTION_MEMORY_TO_PERIPH	Memory to peripheral direction
LL_DMA_DIRECTION_MEMORY_TO_MEMORY	Memory to memory direction

Get Flags Defines

LL_DMA_ISR_GIF1	Channel 1 global flag
LL_DMA_ISR_TCIF1	Channel 1 transfer complete flag
LL_DMA_ISR_HTIF1	Channel 1 half transfer flag
LL_DMA_ISR_TEIF1	Channel 1 transfer error flag
LL_DMA_ISR_GIF2	Channel 2 global flag
LL_DMA_ISR_TCIF2	Channel 2 transfer complete flag
LL_DMA_ISR_HTIF2	Channel 2 half transfer flag
LL_DMA_ISR_TEIF2	Channel 2 transfer error flag
LL_DMA_ISR_GIF3	Channel 3 global flag
LL_DMA_ISR_TCIF3	Channel 3 transfer complete flag
LL_DMA_ISR_HTIF3	Channel 3 half transfer flag
LL_DMA_ISR_TEIF3	Channel 3 transfer error flag
LL_DMA_ISR_GIF4	Channel 4 global flag
LL_DMA_ISR_TCIF4	Channel 4 transfer complete flag
LL_DMA_ISR_HTIF4	Channel 4 half transfer flag
LL_DMA_ISR_TEIF4	Channel 4 transfer error flag
LL_DMA_ISR_GIF5	Channel 5 global flag
LL_DMA_ISR_TCIF5	Channel 5 transfer complete flag
LL_DMA_ISR_HTIF5	Channel 5 half transfer flag
LL_DMA_ISR_TEIF5	Channel 5 transfer error flag
LL_DMA_ISR_GIF6	Channel 6 global flag
LL_DMA_ISR_TCIF6	Channel 6 transfer complete flag
LL_DMA_ISR_HTIF6	Channel 6 half transfer flag

LL_DMA_ISR_TEIF6	Channel 6 transfer error flag
LL_DMA_ISR_GIF7	Channel 7 global flag
LL_DMA_ISR_TCIF7	Channel 7 transfer complete flag
LL_DMA_ISR_HTIF7	Channel 7 half transfer flag
LL_DMA_ISR_TEIF7	Channel 7 transfer error flag

IT Defines

LL_DMA_CCR_TCIE	Transfer complete interrupt
LL_DMA_CCR_HTIE	Half Transfer interrupt
LL_DMA_CCR_TEIE	Transfer error interrupt

Memory data alignment

LL_DMA_MDATAALIGN_BYTE	Memory data alignment : Byte
LL_DMA_MDATAALIGN_HALFWORD	Memory data alignment : HalfWord
LL_DMA_MDATAALIGN_WORD	Memory data alignment : Word

Memory increment mode

LL_DMA_MEMORY_INCREMENT	Memory increment mode Enable
LL_DMA_MEMORY_NOINCREMENT	Memory increment mode Disable

Transfer mode

LL_DMA_MODE_NORMAL	Normal Mode
LL_DMA_MODE_CIRCULAR	Circular Mode

Peripheral data alignment

LL_DMA_PDATAALIGN_BYTE	Peripheral data alignment : Byte
LL_DMA_PDATAALIGN_HALFWORD	Peripheral data alignment : HalfWord
LL_DMA_PDATAALIGN_WORD	Peripheral data alignment : Word

Peripheral increment mode

LL_DMA_PERIPH_INCREMENT	Peripheral increment mode Enable
LL_DMA_PERIPH_NOINCREMENT	Peripheral increment mode Disable

Transfer Priority level

LL_DMA_PRIORITY_LOW	Priority level : Low
LL_DMA_PRIORITY_MEDIUM	Priority level : Medium
LL_DMA_PRIORITY_HIGH	Priority level : High
LL_DMA_PRIORITY_VERYHIGH	Priority level : Very_High

Convert DMAxChannely

__LL_DMA_GET_INSTANCE

Description:

- Convert DMAx_Channely into DMAx.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely

Return value:

- DMAx

__LL_DMA_GET_CHANNEL

Description:

- Convert DMAx_Channely into LL_DMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely

Return value:

- LL_DMA_CHANNEL_y

__LL_DMA_GET_CHANNEL_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_CHANNEL_y into DMAx_Channely.

Parameters:

- __DMA_INSTANCE__: DMAx
- __CHANNEL__: LL_DMA_CHANNEL_y

Return value:

- DMAx_Channely

Common Write and read registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be read

Return value:

- Register: value

52 LL EXTI Generic Driver

52.1 EXTI Firmware driver registers structures

52.1.1 LL_EXTI_InitTypeDef

Data Fields

- ***uint32_t Line_0_31***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

52.2 EXTI Firmware driver API description

52.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name	**__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)**
Function description	Enable ExtiLine Interrupt request for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be one of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13

- LL_EXTI_LINE_14
- LL_EXTI_LINE_15
- LL_EXTI_LINE_16
- LL_EXTI_LINE_17
- LL_EXTI_LINE_18
- LL_EXTI_LINE_19
- LL_EXTI_LINE_ALL_0_31

Return values
- **None:**

Notes
- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- IMR IMx LL_EXTI_EnableIT_0_31

LL_EXTI_DisableIT_0_31

Function name
__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description
Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_ALL_0_31

Return values
- **None:**

Notes
- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line

availability

Reference Manual to LL API cross reference:
- IMR IMx LL_EXTI_DisableIT_0_31

LL_EXTI_IsEnabledIT_0_31

Function name
__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description
Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_ALL_0_31

Return values
- **State:** of bit (1 or 0).

Notes
- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- IMR IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_EnableEvent_0_31

Function name
__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description
Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be one of the following values:

- LL_EXTI_LINE_0
- LL_EXTI_LINE_1
- LL_EXTI_LINE_2
- LL_EXTI_LINE_3
- LL_EXTI_LINE_4
- LL_EXTI_LINE_5
- LL_EXTI_LINE_6
- LL_EXTI_LINE_7
- LL_EXTI_LINE_8
- LL_EXTI_LINE_9
- LL_EXTI_LINE_10
- LL_EXTI_LINE_11
- LL_EXTI_LINE_12
- LL_EXTI_LINE_13
- LL_EXTI_LINE_14
- LL_EXTI_LINE_15
- LL_EXTI_LINE_16
- LL_EXTI_LINE_17
- LL_EXTI_LINE_18
- LL_EXTI_LINE_19
- LL_EXTI_LINE_ALL_0_31

Return values
- **None:**

Notes
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- EMR EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_DisableEvent_0_31

Function name **__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)**

Function description Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15

- LL_EXTI_LINE_16
- LL_EXTI_LINE_17
- LL_EXTI_LINE_18
- LL_EXTI_LINE_19
- LL_EXTI_LINE_ALL_0_31

Return values
- **None:**

Notes
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- EMR EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_IsEnabledEvent_0_31

Function name
__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description
Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_ALL_0_31

Return values
- **State:** of bit (1 or 0).

Notes
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- EMR EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_EnableRisingTrig_0_31

Function name
__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description
Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- RTSR RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_0_31

Function name
__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description
Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3

  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- RTSR RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_0_31

Function name
__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)

Function description
Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15

  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **State:** of bit (1 or 0).

Notes
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- RTSR RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_EnableFallingTrig_0_31

Function name
__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)

Function description
Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **None:**

Notes
- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- FTSR FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_0_31

Function name	**__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3 – LL_EXTI_LINE_4 – LL_EXTI_LINE_5 – LL_EXTI_LINE_6 – LL_EXTI_LINE_7 – LL_EXTI_LINE_8 – LL_EXTI_LINE_9 – LL_EXTI_LINE_10 – LL_EXTI_LINE_11 – LL_EXTI_LINE_12 – LL_EXTI_LINE_13 – LL_EXTI_LINE_14 – LL_EXTI_LINE_15 – LL_EXTI_LINE_16 – LL_EXTI_LINE_18 – LL_EXTI_LINE_19
Return values	• **None:**
Notes	• The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition. • Please check each device line mapping for EXTI Line availability
Reference Manual to LL API cross reference:	• FTSR FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_0_31

Function name	**__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)**
Function description	Check if falling edge trigger is enabled for Lines in range 0 to 31.
Parameters	• **ExtiLine:** This parameter can be a combination of the following values: – LL_EXTI_LINE_0 – LL_EXTI_LINE_1 – LL_EXTI_LINE_2 – LL_EXTI_LINE_3

  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **State:** of bit (1 or 0).

Notes
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- FTSR FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_GenerateSWI_0_31

Function name
__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)

Function description
Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **None:**

Notes
- If the interrupt is enabled on this line in the EXTI_IMR, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR register (by writing a 1 into the bit)
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- SWIER SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_IsActiveFlag_0_31

Function name **__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_0_31 (uint32_t ExtiLine)**

Function description Check if the ExtLine Flag is set or not for Lines in range 0 to 31.

Parameters
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values
- **State:** of bit (1 or 0).

Notes
- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- PR PIFx LL_EXTI_IsActiveFlag_0_31

LL_EXTI_ReadFlag_0_31

Function name **__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)**

Function description: Read ExtLine Combination Flag for Lines in range 0 to 31.

Parameters:
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19

Return values:
- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes:
- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- PR PIFx LL_EXTI_ReadFlag_0_31

LL_EXTI_ClearFlag_0_31

Function name: **__STATIC_INLINE void LL_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)**

Function description: Clear ExtLine Flags for Lines in range 0 to 31.

Parameters:
- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10

- LL_EXTI_LINE_11
- LL_EXTI_LINE_12
- LL_EXTI_LINE_13
- LL_EXTI_LINE_14
- LL_EXTI_LINE_15
- LL_EXTI_LINE_16
- LL_EXTI_LINE_18
- LL_EXTI_LINE_19

Return values
- **None:**

Notes
- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:
- PR PIFx LL_EXTI_ClearFlag_0_31

LL_EXTI_Init

Function name **uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)**

Function description Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.

Parameters
- **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: EXTI registers are initialized
  - ERROR: not applicable

LL_EXTI_DeInit

Function name **uint32_t LL_EXTI_DeInit (void)**

Function description De-initialize the EXTI registers to their default reset values.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: EXTI registers are de-initialized
  - ERROR: not applicable

LL_EXTI_StructInit

Function name **void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)**

Function description Set each LL_EXTI_InitTypeDef field to default value.

Parameters
- **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.

Return values
- **None:**

52.3 EXTI Firmware driver defines

52.3.1 EXTI

LINE

LL_EXTI_LINE_0	Extended line 0
LL_EXTI_LINE_1	Extended line 1
LL_EXTI_LINE_2	Extended line 2
LL_EXTI_LINE_3	Extended line 3
LL_EXTI_LINE_4	Extended line 4
LL_EXTI_LINE_5	Extended line 5
LL_EXTI_LINE_6	Extended line 6
LL_EXTI_LINE_7	Extended line 7
LL_EXTI_LINE_8	Extended line 8
LL_EXTI_LINE_9	Extended line 9
LL_EXTI_LINE_10	Extended line 10
LL_EXTI_LINE_11	Extended line 11
LL_EXTI_LINE_12	Extended line 12
LL_EXTI_LINE_13	Extended line 13
LL_EXTI_LINE_14	Extended line 14
LL_EXTI_LINE_15	Extended line 15
LL_EXTI_LINE_16	Extended line 16
LL_EXTI_LINE_17	Extended line 17
LL_EXTI_LINE_18	Extended line 18
LL_EXTI_LINE_ALL_0_31	All Extended line not reserved
LL_EXTI_LINE_ALL	All Extended line
LL_EXTI_LINE_NONE	None Extended line

Mode

LL_EXTI_MODE_IT	Interrupt Mode
LL_EXTI_MODE_EVENT	Event Mode
LL_EXTI_MODE_IT_EVENT	Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE	No Trigger Mode
LL_EXTI_TRIGGER_RISING	Trigger Rising Mode
LL_EXTI_TRIGGER_FALLING	Trigger Falling Mode
LL_EXTI_TRIGGER_RISING_FALLING	Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg **Description:**

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

53 LL GPIO Generic Driver

53.1 GPIO Firmware driver registers structures

53.1.1 LL_GPIO_InitTypeDef

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***
  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***
  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.

53.2 GPIO Firmware driver API description

53.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name **__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)**

Function description Configure gpio mode for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4

  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_ANALOG
  - LL_GPIO_MODE_FLOATING
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE

Return values
- **None:**

Notes
- I/O mode can be Analog, Floating input, Input with pull-up/pull-down, General purpose Output, Alternate function Output.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- CRL CNFy LL_GPIO_SetPinMode
- CRL MODEy LL_GPIO_SetPinMode
- CRH CNFy LL_GPIO_SetPinMode
- CRH MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio mode for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values
- **Returned:** value can be one of the following values:
  - LL_GPIO_MODE_ANALOG
  - LL_GPIO_MODE_FLOATING
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE

Notes
- I/O mode can be Analog, Floating input, Input with pull-up/pull-down, General purpose Output, Alternate function Output.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- CRL CNFy LL_GPIO_GetPinMode
- CRL MODEy LL_GPIO_GetPinMode
- CRH CNFy LL_GPIO_GetPinMode
- CRH MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinSpeed

Function name
__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)

Function description
Configure gpio speed for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH

Return values
- **None:**

Notes
- I/O speed can be Low, Medium or Fast speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross
- CRL MODEy LL_GPIO_SetPinSpeed
- CRH MODEy LL_GPIO_SetPinSpeed

reference:

LL_GPIO_GetPinSpeed

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio speed for a dedicated pin on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values
- **Returned:** value can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH

Notes
- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:
- CRL MODEy LL_GPIO_GetPinSpeed
- CRH MODEy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinOutputType

Function name
__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t OutputType)

Function description
Configure gpio output type for several pins on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4

  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL
- **OutputType:** This parameter can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Return values
- **None:**

Notes
- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.

Reference Manual to LL API cross reference:
- CRL MODEy LL_GPIO_SetPinOutputType
- CRH MODEy LL_GPIO_SetPinOutputType

LL_GPIO_GetPinOutputType

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio output type for several pins on dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values
- **Returned:** value can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL

– LL_GPIO_OUTPUT_OPENDRAIN

Notes
- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- CRL MODEy LL_GPIO_GetPinOutputType
- CRH MODEy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinPull

Function name
__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)

Function description
Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
- **Pull:** This parameter can be one of the following values:
  – LL_GPIO_PULL_DOWN
  – LL_GPIO_PULL_UP

Return values
- **None:**

Notes
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- ODR ODR LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name
__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description
Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:

  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values
- **Returned:** value can be one of the following values:
  - LL_GPIO_PULL_DOWN
  - LL_GPIO_PULL_UP

Notes
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:
- ODR ODR LL_GPIO_GetPinPull

LL_GPIO_LockPin

Function name

__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock configuration of several pins for a dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values
- **None:**

Notes
- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a specific configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:
- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name
__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description
Return 1 if all pins passed as parameter, of a dedicated port, are locked.

Parameters
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name
__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (GPIO_TypeDef * GPIOx)

Function description
Return 1 if one of the pin of a dedicated port is locked.

Parameters
- **GPIOx:** GPIO Port

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name: **__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (GPIO_TypeDef * GPIOx)**

Function description: Return full input data register value for a dedicated port.

Parameters:
- **GPIOx:** GPIO Port

Return values:
- **Input:** data register value of port

Reference Manual to LL API cross reference:
- IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name: **__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)**

Function description: Return if input data level for several pins of dedicated port is high or low.

Parameters:
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name: **__STATIC_INLINE void LL_GPIO_WriteOutputPort**

(GPIO_TypeDef * GPIOx, uint32_t PortValue)

Function description	Write output data register for the port.
Parameters	• **GPIOx:** GPIO Port • **PortValue:** Level value for each pin of the port
Return values	• **None:**
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name	**__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)**
Function description	Return full output data register value for a dedicated port.
Parameters	• **GPIOx:** GPIO Port
Return values	• **Output:** data register value of port
Reference Manual to LL API cross reference:	• ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name	**__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)**
Function description	Return if input data level for several pins of dedicated port is high or low.
Parameters	• **GPIOx:** GPIO Port • **PinMask:** This parameter can be a combination of the following values: – LL_GPIO_PIN_0 – LL_GPIO_PIN_1 – LL_GPIO_PIN_2 – LL_GPIO_PIN_3 – LL_GPIO_PIN_4 – LL_GPIO_PIN_5 – LL_GPIO_PIN_6 – LL_GPIO_PIN_7 – LL_GPIO_PIN_8 – LL_GPIO_PIN_9 – LL_GPIO_PIN_10 – LL_GPIO_PIN_11 – LL_GPIO_PIN_12 – LL_GPIO_PIN_13 – LL_GPIO_PIN_14 – LL_GPIO_PIN_15 – LL_GPIO_PIN_ALL
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name: **__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**

Function description: Set several pins to high level on dedicated gpio port.

Parameters:
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values:
- **None:**

Reference Manual to LL API cross reference:
- BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name: **__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**

Function description: Set several pins to low level on dedicated gpio port.

Parameters:
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

- LL_GPIO_PIN_8
- LL_GPIO_PIN_9
- LL_GPIO_PIN_10
- LL_GPIO_PIN_11
- LL_GPIO_PIN_12
- LL_GPIO_PIN_13
- LL_GPIO_PIN_14
- LL_GPIO_PIN_15
- LL_GPIO_PIN_ALL

Return values
- **None:**

Reference Manual to LL API cross reference:
- BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name **__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)**

Function description Toggle data value for several pin of dedicated port.

Parameters
- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values
- **None:**

Reference Manual to LL API cross reference:
- ODR ODy LL_GPIO_TogglePin

LL_GPIO_AF_EnableRemap_SPI1

Function name **__STATIC_INLINE void LL_GPIO_AF_EnableRemap_SPI1 (void)**

Function description Enable the remapping of SPI1 alternate function NSS, SCK, MISO

and MOSI.

Return values
- **None:**

Notes
- ENABLE: Remap (NSS/PA15, SCK/PB3, MISO/PB4, MOSI/PB5)

Reference Manual to LL API cross reference:
- MAPR SPI1_REMAP LL_GPIO_AF_EnableRemap_SPI1

LL_GPIO_AF_DisableRemap_SPI1

Function name **__STATIC_INLINE void LL_GPIO_AF_DisableRemap_SPI1 (void)**

Function description Disable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

Return values
- **None:**

Notes
- DISABLE: No remap (NSS/PA4, SCK/PA5, MISO/PA6, MOSI/PA7)

Reference Manual to LL API cross reference:
- MAPR SPI1_REMAP LL_GPIO_AF_DisableRemap_SPI1

LL_GPIO_AF_IsEnabledRemap_SPI1

Function name **__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_SPI1 (void)**

Function description Check if SPI1 has been remaped or not.

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- MAPR SPI1_REMAP LL_GPIO_AF_IsEnabledRemap_SPI1

LL_GPIO_AF_EnableRemap_I2C1

Function name **__STATIC_INLINE void LL_GPIO_AF_EnableRemap_I2C1 (void)**

Function description Enable the remapping of I2C1 alternate function SCL and SDA.

Return values
- **None:**

Notes
- ENABLE: Remap (SCL/PB8, SDA/PB9)

Reference Manual to LL API cross reference:
- MAPR I2C1_REMAP LL_GPIO_AF_EnableRemap_I2C1

LL_GPIO_AF_DisableRemap_I2C1

Function name **__STATIC_INLINE void LL_GPIO_AF_DisableRemap_I2C1 (void)**

Function description	Disable the remapping of I2C1 alternate function SCL and SDA.
Return values	• **None:**
Notes	• DISABLE: No remap (SCL/PB6, SDA/PB7)
Reference Manual to LL API cross reference:	• MAPR I2C1_REMAP LL_GPIO_AF_DisableRemap_I2C1

LL_GPIO_AF_IsEnabledRemap_I2C1

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_I2C1 (void)**
Function description	Check if I2C1 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR I2C1_REMAP LL_GPIO_AF_IsEnabledRemap_I2C1

LL_GPIO_AF_EnableRemap_USART1

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_USART1 (void)**
Function description	Enable the remapping of USART1 alternate function TX and RX.
Return values	• **None:**
Notes	• ENABLE: Remap (TX/PB6, RX/PB7)
Reference Manual to LL API cross reference:	• MAPR USART1_REMAP LL_GPIO_AF_EnableRemap_USART1

LL_GPIO_AF_DisableRemap_USART1

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_USART1 (void)**
Function description	Disable the remapping of USART1 alternate function TX and RX.
Return values	• **None:**
Notes	• DISABLE: No remap (TX/PA9, RX/PA10)
Reference Manual to LL API cross reference:	• MAPR USART1_REMAP LL_GPIO_AF_DisableRemap_USART1

LL_GPIO_AF_IsEnabledRemap_USART1

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_USART1 (void)**
Function description	Check if USART1 has been remaped or not.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• MAPR USART1_REMAP LL_GPIO_AF_IsEnabledRemap_USART1

LL_GPIO_AF_EnableRemap_USART2

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_USART2 (void)**
Function description	Enable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.
Return values	• **None:**
Notes	• ENABLE: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)
Reference Manual to LL API cross reference:	• MAPR USART2_REMAP LL_GPIO_AF_EnableRemap_USART2

LL_GPIO_AF_DisableRemap_USART2

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_USART2 (void)**
Function description	Disable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.
Return values	• **None:**
Notes	• DISABLE: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)
Reference Manual to LL API cross reference:	• MAPR USART2_REMAP LL_GPIO_AF_DisableRemap_USART2

LL_GPIO_AF_IsEnabledRemap_USART2

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_USART2 (void)**
Function description	Check if USART2 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR USART2_REMAP LL_GPIO_AF_IsEnabledRemap_USART2

LL_GPIO_AF_EnableRemap_USART3

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_USART3 (void)**
Function description	Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.
Return values	• **None:**

Notes	• ENABLE: Full remap (TX/PD8, RX/PD9, CK/PD10, CTS/PD11, RTS/PD12)
Reference Manual to LL API cross reference:	• MAPR USART3_REMAP LL_GPIO_AF_EnableRemap_USART3

LL_GPIO_AF_RemapPartial_USART3

Function name	**__STATIC_INLINE void LL_GPIO_AF_RemapPartial_USART3 (void)**
Function description	Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.
Return values	• **None:**
Notes	• PARTIAL: Partial remap (TX/PC10, RX/PC11, CK/PC12, CTS/PB13, RTS/PB14)
Reference Manual to LL API cross reference:	• MAPR USART3_REMAP LL_GPIO_AF_RemapPartial_USART3

LL_GPIO_AF_DisableRemap_USART3

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_USART3 (void)**
Function description	Disable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.
Return values	• **None:**
Notes	• DISABLE: No remap (TX/PB10, RX/PB11, CK/PB12, CTS/PB13, RTS/PB14)
Reference Manual to LL API cross reference:	• MAPR USART3_REMAP LL_GPIO_AF_DisableRemap_USART3

LL_GPIO_AF_EnableRemap_TIM1

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM1 (void)**
Function description	Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)
Return values	• **None:**
Notes	• ENABLE: Full remap (ETR/PE7, CH1/PE9, CH2/PE11, CH3/PE13, CH4/PE14, BKIN/PE15, CH1N/PE8, CH2N/PE10, CH3N/PE12)
Reference Manual to LL API cross reference:	• MAPR TIM1_REMAP LL_GPIO_AF_EnableRemap_TIM1

LL_GPIO_AF_RemapPartial_TIM1

Function name	**__STATIC_INLINE void LL_GPIO_AF_RemapPartial_TIM1 (void)**
Function description	Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)
Return values	• **None:**
Notes	• PARTIAL: Partial remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PA6, CH1N/PA7, CH2N/PB0, CH3N/PB1)
Reference Manual to LL API cross reference:	• MAPR TIM1_REMAP LL_GPIO_AF_RemapPartial_TIM1

LL_GPIO_AF_DisableRemap_TIM1

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM1 (void)**
Function description	Disable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)
Return values	• **None:**
Notes	• DISABLE: No remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PB12, CH1N/PB13, CH2N/PB14, CH3N/PB15)
Reference Manual to LL API cross reference:	• MAPR TIM1_REMAP LL_GPIO_AF_DisableRemap_TIM1

LL_GPIO_AF_EnableRemap_TIM2

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM2 (void)**
Function description	Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)
Return values	• **None:**
Notes	• ENABLE: Full remap (CH1/ETR/PA15, CH2/PB3, CH3/PB10, CH4/PB11)
Reference Manual to LL API cross reference:	• MAPR TIM2_REMAP LL_GPIO_AF_EnableRemap_TIM2

LL_GPIO_AF_RemapPartial2_TIM2

Function name	**__STATIC_INLINE void LL_GPIO_AF_RemapPartial2_TIM2 (void)**
Function description	Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return values	• **None:**
Notes	• PARTIAL_2: Partial remap (CH1/ETR/PA0, CH2/PA1, CH3/PB10, CH4/PB11)
Reference Manual to LL API cross reference:	• MAPR TIM2_REMAP LL_GPIO_AF_RemapPartial2_TIM2

LL_GPIO_AF_RemapPartial1_TIM2

Function name	**__STATIC_INLINE void LL_GPIO_AF_RemapPartial1_TIM2 (void)**
Function description	Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)
Return values	• **None:**
Notes	• PARTIAL_1: Partial remap (CH1/ETR/PA15, CH2/PB3, CH3/PA2, CH4/PA3)
Reference Manual to LL API cross reference:	• MAPR TIM2_REMAP LL_GPIO_AF_RemapPartial1_TIM2

LL_GPIO_AF_DisableRemap_TIM2

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM2 (void)**
Function description	Disable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)
Return values	• **None:**
Notes	• DISABLE: No remap (CH1/ETR/PA0, CH2/PA1, CH3/PA2, CH4/PA3)
Reference Manual to LL API cross reference:	• MAPR TIM2_REMAP LL_GPIO_AF_DisableRemap_TIM2

LL_GPIO_AF_EnableRemap_TIM3

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM3 (void)**
Function description	Enable the remapping of TIM3 alternate function channels 1 to 4.
Return values	• **None:**
Notes	• ENABLE: Full remap (CH1/PC6, CH2/PC7, CH3/PC8, CH4/PC9) • TIM3_ETR on PE0 is not re-mapped.
Reference Manual to LL API cross reference:	• MAPR TIM3_REMAP LL_GPIO_AF_EnableRemap_TIM3

LL_GPIO_AF_RemapPartial_TIM3

Function name	**__STATIC_INLINE void LL_GPIO_AF_RemapPartial_TIM3 (void)**
Function description	Enable the remapping of TIM3 alternate function channels 1 to 4.
Return values	• **None:**
Notes	• PARTIAL: Partial remap (CH1/PB4, CH2/PB5, CH3/PB0, CH4/PB1) • TIM3_ETR on PE0 is not re-mapped.
Reference Manual to LL API cross reference:	• MAPR TIM3_REMAP LL_GPIO_AF_RemapPartial_TIM3

LL_GPIO_AF_DisableRemap_TIM3

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM3 (void)**
Function description	Disable the remapping of TIM3 alternate function channels 1 to 4.
Return values	• **None:**
Notes	• DISABLE: No remap (CH1/PA6, CH2/PA7, CH3/PB0, CH4/PB1) • TIM3_ETR on PE0 is not re-mapped.
Reference Manual to LL API cross reference:	• MAPR TIM3_REMAP LL_GPIO_AF_DisableRemap_TIM3

LL_GPIO_AF_EnableRemap_TIM4

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM4 (void)**
Function description	Enable the remapping of TIM4 alternate function channels 1 to 4.
Return values	• **None:**
Notes	• ENABLE: Full remap (TIM4_CH1/PD12, TIM4_CH2/PD13, TIM4_CH3/PD14, TIM4_CH4/PD15) • TIM4_ETR on PE0 is not re-mapped.
Reference Manual to LL API cross reference:	• MAPR TIM4_REMAP LL_GPIO_AF_EnableRemap_TIM4

LL_GPIO_AF_DisableRemap_TIM4

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM4 (void)**
Function description	Disable the remapping of TIM4 alternate function channels 1 to 4.
Return values	• **None:**
Notes	• DISABLE: No remap (TIM4_CH1/PB6, TIM4_CH2/PB7, TIM4_CH3/PB8, TIM4_CH4/PB9)

- TIM4_ETR on PE0 is not re-mapped.

Reference Manual to LL API cross reference:

- MAPR TIM4_REMAP LL_GPIO_AF_DisableRemap_TIM4

LL_GPIO_AF_IsEnabledRemap_TIM4

Function name **__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM4 (void)**

Function description Check if TIM4 has been remaped or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MAPR TIM4_REMAP LL_GPIO_AF_IsEnabledRemap_TIM4

LL_GPIO_AF_RemapPartial1_CAN1

Function name **__STATIC_INLINE void LL_GPIO_AF_RemapPartial1_CAN1 (void)**

Function description Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return values

- **None:**

Notes

- CASE 1: CAN_RX mapped to PA11, CAN_TX mapped to PA12

Reference Manual to LL API cross reference:

- MAPR CAN_REMAP LL_GPIO_AF_RemapPartial1_CAN1

LL_GPIO_AF_RemapPartial2_CAN1

Function name **__STATIC_INLINE void LL_GPIO_AF_RemapPartial2_CAN1 (void)**

Function description Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return values

- **None:**

Notes

- CASE 2: CAN_RX mapped to PB8, CAN_TX mapped to PB9 (not available on 36-pin package)

Reference Manual to LL API cross reference:

- MAPR CAN_REMAP LL_GPIO_AF_RemapPartial2_CAN1

LL_GPIO_AF_RemapPartial3_CAN1

Function name **__STATIC_INLINE void LL_GPIO_AF_RemapPartial3_CAN1 (void)**

Function description Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return values	• **None:**
Notes	• CASE 3: CAN_RX mapped to PD0, CAN_TX mapped to PD1
Reference Manual to LL API cross reference:	• MAPR CAN_REMAP LL_GPIO_AF_RemapPartial3_CAN1

LL_GPIO_AF_EnableRemap_PD01

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_PD01 (void)**
Function description	Enable the remapping of PD0 and PD1.
Return values	• **None:**
Notes	• ENABLE: PD0 remapped on OSC_IN, PD1 remapped on OSC_OUT.
Reference Manual to LL API cross reference:	• MAPR PD01_REMAP LL_GPIO_AF_EnableRemap_PD01

LL_GPIO_AF_DisableRemap_PD01

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_PD01 (void)**
Function description	Disable the remapping of PD0 and PD1.
Return values	• **None:**
Notes	• DISABLE: No remapping of PD0 and PD1
Reference Manual to LL API cross reference:	• MAPR PD01_REMAP LL_GPIO_AF_DisableRemap_PD01

LL_GPIO_AF_IsEnabledRemap_PD01

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_PD01 (void)**
Function description	Check if PD01 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR PD01_REMAP LL_GPIO_AF_IsEnabledRemap_PD01

LL_GPIO_AF_EnableRemap_TIM5CH4

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM5CH4 (void)**
Function description	Enable the remapping of TIM5CH4.
Return values	• **None:**

Notes	• ENABLE: LSI internal clock is connected to TIM5_CH4 input for calibration purpose. • This function is available only in high density value line devices.
Reference Manual to LL API cross reference:	• MAPR TIM5CH4_IREMAP LL_GPIO_AF_EnableRemap_TIM5CH4

LL_GPIO_AF_DisableRemap_TIM5CH4

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM5CH4 (void)**
Function description	Disable the remapping of TIM5CH4.
Return values	• **None:**
Notes	• DISABLE: TIM5_CH4 is connected to PA3 • This function is available only in high density value line devices.
Reference Manual to LL API cross reference:	• MAPR TIM5CH4_IREMAP LL_GPIO_AF_DisableRemap_TIM5CH4

LL_GPIO_AF_IsEnabledRemap_TIM5CH4

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM5CH4 (void)**
Function description	Check if TIM5CH4 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR TIM5CH4_IREMAP LL_GPIO_AF_IsEnabledRemap_TIM5CH4

LL_GPIO_AF_EnableRemap_ADC1_ETRGINJ

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_ADC1_ETRGINJ (void)**
Function description	Enable the remapping of ADC1_ETRGINJ (ADC 1 External trigger injected conversion).
Return values	• **None:**
Notes	• ENABLE: ADC1 External Event injected conversion is connected to TIM8 Channel4.
Reference Manual to LL API cross reference:	• MAPR ADC1_ETRGINJ_REMAP LL_GPIO_AF_EnableRemap_ADC1_ETRGINJ

LL_GPIO_AF_DisableRemap_ADC1_ETRGINJ

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_ADC1_ETRGINJ (void)**

Function description	Disable the remapping of ADC1_ETRGINJ (ADC 1 External trigger injected conversion).
Return values	• **None:**
Notes	• DISABLE: ADC1 External trigger injected conversion is connected to EXTI15
Reference Manual to LL API cross reference:	• MAPR ADC1_ETRGINJ_REMAP LL_GPIO_AF_DisableRemap_ADC1_ETRGINJ

LL_GPIO_AF_IsEnabledRemap_ADC1_ETRGINJ

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_ADC1_ETRGINJ (void)**
Function description	Check if ADC1_ETRGINJ has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR ADC1_ETRGINJ_REMAP LL_GPIO_AF_IsEnabledRemap_ADC1_ETRGINJ

LL_GPIO_AF_EnableRemap_ADC1_ETRGREG

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_ADC1_ETRGREG (void)**
Function description	Enable the remapping of ADC1_ETRGREG (ADC 1 External trigger regular conversion).
Return values	• **None:**
Notes	• ENABLE: ADC1 External Event regular conversion is connected to TIM8 TRG0.
Reference Manual to LL API cross reference:	• MAPR ADC1_ETRGREG_REMAP LL_GPIO_AF_EnableRemap_ADC1_ETRGREG

LL_GPIO_AF_DisableRemap_ADC1_ETRGREG

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_ADC1_ETRGREG (void)**
Function description	Disable the remapping of ADC1_ETRGREG (ADC 1 External trigger regular conversion).
Return values	• **None:**
Notes	• DISABLE: ADC1 External trigger regular conversion is connected to EXTI11
Reference Manual to LL API cross reference:	• MAPR ADC1_ETRGREG_REMAP LL_GPIO_AF_DisableRemap_ADC1_ETRGREG

LL_GPIO_AF_IsEnabledRemap_ADC1_ETRGREG

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_ADC1_ETRGREG (void)**
Function description	Check if ADC1_ETRGREG has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR ADC1_ETRGREG_REMAP LL_GPIO_AF_IsEnabledRemap_ADC1_ETRGREG

LL_GPIO_AF_EnableRemap_ADC2_ETRGINJ

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_ADC2_ETRGINJ (void)**
Function description	Enable the remapping of ADC2_ETRGREG (ADC 2 External trigger injected conversion).
Return values	• **None:**
Notes	• ENABLE: ADC2 External Event injected conversion is connected to TIM8 Channel4.
Reference Manual to LL API cross reference:	• MAPR ADC2_ETRGINJ_REMAP LL_GPIO_AF_EnableRemap_ADC2_ETRGINJ

LL_GPIO_AF_DisableRemap_ADC2_ETRGINJ

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_ADC2_ETRGINJ (void)**
Function description	Disable the remapping of ADC2_ETRGREG (ADC 2 External trigger injected conversion).
Return values	• **None:**
Notes	• DISABLE: ADC2 External trigger injected conversion is connected to EXTI15
Reference Manual to LL API cross reference:	• MAPR ADC2_ETRGINJ_REMAP LL_GPIO_AF_DisableRemap_ADC2_ETRGINJ

LL_GPIO_AF_IsEnabledRemap_ADC2_ETRGINJ

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_ADC2_ETRGINJ (void)**
Function description	Check if ADC2_ETRGINJ has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR ADC2_ETRGINJ_REMAP LL_GPIO_AF_IsEnabledRemap_ADC2_ETRGINJ

LL_GPIO_AF_EnableRemap_ADC2_ETRGREG

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_ADC2_ETRGREG (void)**
Function description	Enable the remapping of ADC2_ETRGREG (ADC 2 External trigger regular conversion).
Return values	• **None:**
Notes	• ENABLE: ADC2 External Event regular conversion is connected to TIM8 TRG0.
Reference Manual to LL API cross reference:	• MAPR ADC2_ETRGREG_REMAP LL_GPIO_AF_EnableRemap_ADC2_ETRGREG

LL_GPIO_AF_DisableRemap_ADC2_ETRGREG

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_ADC2_ETRGREG (void)**
Function description	Disable the remapping of ADC2_ETRGREG (ADC 2 External trigger regular conversion).
Return values	• **None:**
Notes	• DISABLE: ADC2 External trigger regular conversion is connected to EXTI11
Reference Manual to LL API cross reference:	• MAPR ADC2_ETRGREG_REMAP LL_GPIO_AF_DisableRemap_ADC2_ETRGREG

LL_GPIO_AF_IsEnabledRemap_ADC2_ETRGREG

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_ADC2_ETRGREG (void)**
Function description	Check if ADC2_ETRGREG has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR ADC2_ETRGREG_REMAP LL_GPIO_AF_IsEnabledRemap_ADC2_ETRGREG

LL_GPIO_AF_EnableRemap_SWJ

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_SWJ (void)**
Function description	Enable the Serial wire JTAG configuration.
Return values	• **None:**
Notes	• ENABLE: Full SWJ (JTAG-DP + SW-DP): Reset State
Reference Manual to LL API cross reference:	• MAPR SWJ_CFG LL_GPIO_AF_EnableRemap_SWJ

LL_GPIO_AF_Remap_SWJ_NONJTRST

Function name	**__STATIC_INLINE void LL_GPIO_AF_Remap_SWJ_NONJTRST (void)**
Function description	Enable the Serial wire JTAG configuration.
Return values	• **None:**
Notes	• NONJTRST: Full SWJ (JTAG-DP + SW-DP) but without NJTRST
Reference Manual to LL API cross reference:	• MAPR SWJ_CFG LL_GPIO_AF_Remap_SWJ_NONJTRST

LL_GPIO_AF_Remap_SWJ_NOJTAG

Function name	**__STATIC_INLINE void LL_GPIO_AF_Remap_SWJ_NOJTAG (void)**
Function description	Enable the Serial wire JTAG configuration.
Return values	• **None:**
Notes	• NOJTAG: JTAG-DP Disabled and SW-DP Enabled
Reference Manual to LL API cross reference:	• MAPR SWJ_CFG LL_GPIO_AF_Remap_SWJ_NOJTAG

LL_GPIO_AF_DisableRemap_SWJ

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_SWJ (void)**
Function description	Disable the Serial wire JTAG configuration.
Return values	• **None:**
Notes	• DISABLE: JTAG-DP Disabled and SW-DP Disabled
Reference Manual to LL API cross reference:	• MAPR SWJ_CFG LL_GPIO_AF_DisableRemap_SWJ

LL_GPIO_AF_EnableRemap_TIM9

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM9 (void)**
Function description	Enable the remapping of TIM9_CH1 and TIM9_CH2.
Return values	• **None:**
Notes	• ENABLE: Remap (TIM9_CH1 on PE5 and TIM9_CH2 on PE6).
Reference Manual to LL API cross reference:	• MAPR2 TIM9_REMAP LL_GPIO_AF_EnableRemap_TIM9

LL_GPIO_AF_DisableRemap_TIM9

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM9 (void)**
Function description	Disable the remapping of TIM9_CH1 and TIM9_CH2.
Return values	• **None:**
Notes	• DISABLE: No remap (TIM9_CH1 on PA2 and TIM9_CH2 on PA3).
Reference Manual to LL API cross reference:	• MAPR2 TIM9_REMAP LL_GPIO_AF_DisableRemap_TIM9

LL_GPIO_AF_IsEnabledRemap_TIM9

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM9 (void)**
Function description	Check if TIM9_CH1 and TIM9_CH2 have been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR2 TIM9_REMAP LL_GPIO_AF_IsEnabledRemap_TIM9

LL_GPIO_AF_EnableRemap_TIM10

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM10 (void)**
Function description	Enable the remapping of TIM10_CH1.
Return values	• **None:**
Notes	• ENABLE: Remap (TIM10_CH1 on PF6).
Reference Manual to LL API cross reference:	• MAPR2 TIM10_REMAP LL_GPIO_AF_EnableRemap_TIM10

LL_GPIO_AF_DisableRemap_TIM10

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM10 (void)**
Function description	Disable the remapping of TIM10_CH1.
Return values	• **None:**
Notes	• DISABLE: No remap (TIM10_CH1 on PB8).
Reference Manual to LL API cross reference:	• MAPR2 TIM10_REMAP LL_GPIO_AF_DisableRemap_TIM10

LL_GPIO_AF_IsEnabledRemap_TIM10

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM10 (void)**
Function description	Check if TIM10_CH1 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR2 TIM10_REMAP LL_GPIO_AF_IsEnabledRemap_TIM10

LL_GPIO_AF_EnableRemap_TIM11

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM11 (void)**
Function description	Enable the remapping of TIM11_CH1.
Return values	• **None:**
Notes	• ENABLE: Remap (TIM11_CH1 on PF7).
Reference Manual to LL API cross reference:	• MAPR2 TIM11_REMAP LL_GPIO_AF_EnableRemap_TIM11

LL_GPIO_AF_DisableRemap_TIM11

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM11 (void)**
Function description	Disable the remapping of TIM11_CH1.
Return values	• **None:**
Notes	• DISABLE: No remap (TIM11_CH1 on PB9).
Reference Manual to LL API cross reference:	• MAPR2 TIM11_REMAP LL_GPIO_AF_DisableRemap_TIM11

LL_GPIO_AF_IsEnabledRemap_TIM11

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM11 (void)**
Function description	Check if TIM11_CH1 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR2 TIM11_REMAP LL_GPIO_AF_IsEnabledRemap_TIM11

LL_GPIO_AF_EnableRemap_TIM13

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM13 (void)**

Function description	Enable the remapping of TIM13_CH1.
Return values	• **None:**
Notes	• ENABLE: Remap STM32F100:(TIM13_CH1 on PF8). Others:(TIM13_CH1 on PB0).
Reference Manual to LL API cross reference:	• MAPR2 TIM13_REMAP LL_GPIO_AF_EnableRemap_TIM13

LL_GPIO_AF_DisableRemap_TIM13

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM13 (void)**
Function description	Disable the remapping of TIM13_CH1.
Return values	• **None:**
Notes	• DISABLE: No remap STM32F100:(TIM13_CH1 on PA6). Others:(TIM13_CH1 on PC8).
Reference Manual to LL API cross reference:	• MAPR2 TIM13_REMAP LL_GPIO_AF_DisableRemap_TIM13

LL_GPIO_AF_IsEnabledRemap_TIM13

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM13 (void)**
Function description	Check if TIM13_CH1 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR2 TIM13_REMAP LL_GPIO_AF_IsEnabledRemap_TIM13

LL_GPIO_AF_EnableRemap_TIM14

Function name	**__STATIC_INLINE void LL_GPIO_AF_EnableRemap_TIM14 (void)**
Function description	Enable the remapping of TIM14_CH1.
Return values	• **None:**
Notes	• ENABLE: Remap STM32F100:(TIM14_CH1 on PB1). Others:(TIM14_CH1 on PF9).
Reference Manual to LL API cross reference:	• MAPR2 TIM14_REMAP LL_GPIO_AF_EnableRemap_TIM14

LL_GPIO_AF_DisableRemap_TIM14

Function name	**__STATIC_INLINE void LL_GPIO_AF_DisableRemap_TIM14 (void)**

Function description	Disable the remapping of TIM14_CH1.
Return values	• **None:**
Notes	• DISABLE: No remap STM32F100:(TIM14_CH1 on PC9). Others:(TIM14_CH1 on PA7).
Reference Manual to LL API cross reference:	• MAPR2 TIM14_REMAP LL_GPIO_AF_DisableRemap_TIM14

LL_GPIO_AF_IsEnabledRemap_TIM14

Function name	**__STATIC_INLINE uint32_t LL_GPIO_AF_IsEnabledRemap_TIM14 (void)**
Function description	Check if TIM14_CH1 has been remaped or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• MAPR2 TIM14_REMAP LL_GPIO_AF_IsEnabledRemap_TIM14

LL_GPIO_AF_Disconnect_FSMCNADV

Function name	**__STATIC_INLINE void LL_GPIO_AF_Disconnect_FSMCNADV (void)**
Function description	Controls the use of the optional FSMC_NADV signal.
Return values	• **None:**
Notes	• DISCONNECTED: The NADV signal is not connected. The I/O pin can be used by another peripheral.
Reference Manual to LL API cross reference:	• MAPR2 FSMC_NADV LL_GPIO_AF_Disconnect_FSMCNADV

LL_GPIO_AF_Connect_FSMCNADV

Function name	**__STATIC_INLINE void LL_GPIO_AF_Connect_FSMCNADV (void)**
Function description	Controls the use of the optional FSMC_NADV signal.
Return values	• **None:**
Notes	• CONNECTED: The NADV signal is connected to the output (default).
Reference Manual to LL API cross reference:	• MAPR2 FSMC_NADV LL_GPIO_AF_Connect_FSMCNADV

LL_GPIO_AF_ConfigEventout

Function name	**__STATIC_INLINE void LL_GPIO_AF_ConfigEventout (uint32_t LL_GPIO_PortSource, uint32_t LL_GPIO_PinSource)**

Function description: Configures the port and pin on which the EVENTOUT Cortex signal will be connected.

Parameters:

- **LL_GPIO_PortSource:** This parameter can be one of the following values:
  - LL_GPIO_AF_EVENTOUT_PORT_A
  - LL_GPIO_AF_EVENTOUT_PORT_B
  - LL_GPIO_AF_EVENTOUT_PORT_C
  - LL_GPIO_AF_EVENTOUT_PORT_D
  - LL_GPIO_AF_EVENTOUT_PORT_E
- **LL_GPIO_PinSource:** This parameter can be one of the following values:
  - LL_GPIO_AF_EVENTOUT_PIN_0
  - LL_GPIO_AF_EVENTOUT_PIN_1
  - LL_GPIO_AF_EVENTOUT_PIN_2
  - LL_GPIO_AF_EVENTOUT_PIN_3
  - LL_GPIO_AF_EVENTOUT_PIN_4
  - LL_GPIO_AF_EVENTOUT_PIN_5
  - LL_GPIO_AF_EVENTOUT_PIN_6
  - LL_GPIO_AF_EVENTOUT_PIN_7
  - LL_GPIO_AF_EVENTOUT_PIN_8
  - LL_GPIO_AF_EVENTOUT_PIN_9
  - LL_GPIO_AF_EVENTOUT_PIN_10
  - LL_GPIO_AF_EVENTOUT_PIN_11
  - LL_GPIO_AF_EVENTOUT_PIN_12
  - LL_GPIO_AF_EVENTOUT_PIN_13
  - LL_GPIO_AF_EVENTOUT_PIN_14
  - LL_GPIO_AF_EVENTOUT_PIN_15

Return values:

- **None:**

Reference Manual to LL API cross reference:

- EVCR PORT LL_GPIO_AF_ConfigEventout
- EVCR PIN LL_GPIO_AF_ConfigEventout

LL_GPIO_AF_EnableEventout

Function name: **__STATIC_INLINE void LL_GPIO_AF_EnableEventout (void)**

Function description: Enables the Event Output.

Return values:

- **None:**

Reference Manual to LL API cross reference:

- EVCR EVOE LL_GPIO_AF_EnableEventout

LL_GPIO_AF_DisableEventout

Function name: **__STATIC_INLINE void LL_GPIO_AF_DisableEventout (void)**

Function description: Disables the Event Output.

Return values:

- **None:**

Reference Manual to LL API cross

- EVCR EVOE LL_GPIO_AF_DisableEventout

reference:

LL_GPIO_AF_SetEXTISource

Function name **__STATIC_INLINE void LL_GPIO_AF_SetEXTISource (uint32_t Port, uint32_t Line)**

Function description Configure source input for the EXTI external interrupt.

Parameters
- **Port:** This parameter can be one of the following values:
  - LL_GPIO_AF_EXTI_PORTA
  - LL_GPIO_AF_EXTI_PORTB
  - LL_GPIO_AF_EXTI_PORTC
  - LL_GPIO_AF_EXTI_PORTD
  - LL_GPIO_AF_EXTI_PORTE
  - LL_GPIO_AF_EXTI_PORTF
  - LL_GPIO_AF_EXTI_PORTG
- **Line:** This parameter can be one of the following values:
  - LL_GPIO_AF_EXTI_LINE0
  - LL_GPIO_AF_EXTI_LINE1
  - LL_GPIO_AF_EXTI_LINE2
  - LL_GPIO_AF_EXTI_LINE3
  - LL_GPIO_AF_EXTI_LINE4
  - LL_GPIO_AF_EXTI_LINE5
  - LL_GPIO_AF_EXTI_LINE6
  - LL_GPIO_AF_EXTI_LINE7
  - LL_GPIO_AF_EXTI_LINE8
  - LL_GPIO_AF_EXTI_LINE9
  - LL_GPIO_AF_EXTI_LINE10
  - LL_GPIO_AF_EXTI_LINE11
  - LL_GPIO_AF_EXTI_LINE12
  - LL_GPIO_AF_EXTI_LINE13
  - LL_GPIO_AF_EXTI_LINE14
  - LL_GPIO_AF_EXTI_LINE15

Return values
- **None:**

Reference Manual to LL API cross reference:
- AFIO_EXTICR1 EXTIx LL_GPIO_AF_SetEXTISource
- AFIO_EXTICR2 EXTIx LL_GPIO_AF_SetEXTISource
- AFIO_EXTICR3 EXTIx LL_GPIO_AF_SetEXTISource
- AFIO_EXTICR4 EXTIx LL_GPIO_AF_SetEXTISource

LL_GPIO_AF_GetEXTISource

Function name **__STATIC_INLINE uint32_t LL_GPIO_AF_GetEXTISource (uint32_t Line)**

Function description Get the configured defined for specific EXTI Line.

Parameters
- **Line:** This parameter can be one of the following values:
  - LL_GPIO_AF_EXTI_LINE0
  - LL_GPIO_AF_EXTI_LINE1
  - LL_GPIO_AF_EXTI_LINE2
  - LL_GPIO_AF_EXTI_LINE3
  - LL_GPIO_AF_EXTI_LINE4
  - LL_GPIO_AF_EXTI_LINE5

  - LL_GPIO_AF_EXTI_LINE6
  - LL_GPIO_AF_EXTI_LINE7
  - LL_GPIO_AF_EXTI_LINE8
  - LL_GPIO_AF_EXTI_LINE9
  - LL_GPIO_AF_EXTI_LINE10
  - LL_GPIO_AF_EXTI_LINE11
  - LL_GPIO_AF_EXTI_LINE12
  - LL_GPIO_AF_EXTI_LINE13
  - LL_GPIO_AF_EXTI_LINE14
  - LL_GPIO_AF_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_EXTI_PORTA
  - LL_GPIO_AF_EXTI_PORTB
  - LL_GPIO_AF_EXTI_PORTC
  - LL_GPIO_AF_EXTI_PORTD
  - LL_GPIO_AF_EXTI_PORTE
  - LL_GPIO_AF_EXTI_PORTF
  - LL_GPIO_AF_EXTI_PORTG

Reference Manual to LL API cross reference:

- AFIO_EXTICR1 EXTIx LL_GPIO_AF_GetEXTISource
- AFIO_EXTICR2 EXTIx LL_GPIO_AF_GetEXTISource
- AFIO_EXTICR3 EXTIx LL_GPIO_AF_GetEXTISource
- AFIO_EXTICR4 EXTIx LL_GPIO_AF_GetEXTISource

LL_GPIO_DeInit

Function name

ErrorStatus LL_GPIO_DeInit (GPIO_TypeDef * GPIOx)

Function description

De-initialize GPIO registers (Registers restored to their default values).

Parameters

- **GPIOx:** GPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are de-initialized
  - ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name

ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content
  - ERROR: Not applicable

LL_GPIO_StructInit

Function name	**void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)**
Function description	Set each LL_GPIO_InitTypeDef field to default value.
Parameters	• **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

53.3 GPIO Firmware driver defines

53.3.1 GPIO

GPIO EXTI LINE

LL_GPIO_AF_EXTI_LINE0	EXTI_POSITION_0 | EXTICR[0]
LL_GPIO_AF_EXTI_LINE1	EXTI_POSITION_4 | EXTICR[0]
LL_GPIO_AF_EXTI_LINE2	EXTI_POSITION_8 | EXTICR[0]
LL_GPIO_AF_EXTI_LINE3	EXTI_POSITION_12 | EXTICR[0]
LL_GPIO_AF_EXTI_LINE4	EXTI_POSITION_0 | EXTICR[1]
LL_GPIO_AF_EXTI_LINE5	EXTI_POSITION_4 | EXTICR[1]
LL_GPIO_AF_EXTI_LINE6	EXTI_POSITION_8 | EXTICR[1]
LL_GPIO_AF_EXTI_LINE7	EXTI_POSITION_12 | EXTICR[1]
LL_GPIO_AF_EXTI_LINE8	EXTI_POSITION_0 | EXTICR[2]
LL_GPIO_AF_EXTI_LINE9	EXTI_POSITION_4 | EXTICR[2]
LL_GPIO_AF_EXTI_LINE10	EXTI_POSITION_8 | EXTICR[2]
LL_GPIO_AF_EXTI_LINE11	EXTI_POSITION_12 | EXTICR[2]
LL_GPIO_AF_EXTI_LINE12	EXTI_POSITION_0 | EXTICR[3]
LL_GPIO_AF_EXTI_LINE13	EXTI_POSITION_4 | EXTICR[3]
LL_GPIO_AF_EXTI_LINE14	EXTI_POSITION_8 | EXTICR[3]
LL_GPIO_AF_EXTI_LINE15	EXTI_POSITION_12 | EXTICR[3]

GPIO EXTI PORT

LL_GPIO_AF_EXTI_PORTA	EXTI PORT A
LL_GPIO_AF_EXTI_PORTB	EXTI PORT B
LL_GPIO_AF_EXTI_PORTC	EXTI PORT C
LL_GPIO_AF_EXTI_PORTD	EXTI PORT D
LL_GPIO_AF_EXTI_PORTE	EXTI PORT E
LL_GPIO_AF_EXTI_PORTF	EXTI PORT F
LL_GPIO_AF_EXTI_PORTG	EXTI PORT G

Mode

LL_GPIO_MODE_ANALOG	Select analog mode

LL_GPIO_MODE_FLOATING	Select floating mode
LL_GPIO_MODE_INPUT	Select input mode
LL_GPIO_MODE_OUTPUT	Select general purpose output mode
LL_GPIO_MODE_ALTERNATE	Select alternate function mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL	Select push-pull as output type
LL_GPIO_OUTPUT_OPENDRAIN	Select open-drain as output type

PIN

LL_GPIO_PIN_0	Select pin 0
LL_GPIO_PIN_1	Select pin 1
LL_GPIO_PIN_2	Select pin 2
LL_GPIO_PIN_3	Select pin 3
LL_GPIO_PIN_4	Select pin 4
LL_GPIO_PIN_5	Select pin 5
LL_GPIO_PIN_6	Select pin 6
LL_GPIO_PIN_7	Select pin 7
LL_GPIO_PIN_8	Select pin 8
LL_GPIO_PIN_9	Select pin 9
LL_GPIO_PIN_10	Select pin 10
LL_GPIO_PIN_11	Select pin 11
LL_GPIO_PIN_12	Select pin 12
LL_GPIO_PIN_13	Select pin 13
LL_GPIO_PIN_14	Select pin 14
LL_GPIO_PIN_15	Select pin 15
LL_GPIO_PIN_ALL	Select all pins

Pull Up Pull Down

LL_GPIO_PULL_DOWN	Select I/O pull down
LL_GPIO_PULL_UP	Select I/O pull up

Output Speed

LL_GPIO_MODE_OUTPUT_10MHz	Select Output mode, max speed 10 MHz
LL_GPIO_MODE_OUTPUT_2MHz	Select Output mode, max speed 20 MHz
LL_GPIO_MODE_OUTPUT_50MHz	Select Output mode, max speed 50 MHz

Common Write and read registers Macros

LL_GPIO_WriteReg **Description:**

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

EVENTOUT Pin

LL_GPIO_AF_EVENTOUT_PIN_0	EVENTOUT on pin 0
LL_GPIO_AF_EVENTOUT_PIN_1	EVENTOUT on pin 1
LL_GPIO_AF_EVENTOUT_PIN_2	EVENTOUT on pin 2
LL_GPIO_AF_EVENTOUT_PIN_3	EVENTOUT on pin 3
LL_GPIO_AF_EVENTOUT_PIN_4	EVENTOUT on pin 4
LL_GPIO_AF_EVENTOUT_PIN_5	EVENTOUT on pin 5
LL_GPIO_AF_EVENTOUT_PIN_6	EVENTOUT on pin 6
LL_GPIO_AF_EVENTOUT_PIN_7	EVENTOUT on pin 7
LL_GPIO_AF_EVENTOUT_PIN_8	EVENTOUT on pin 8
LL_GPIO_AF_EVENTOUT_PIN_9	EVENTOUT on pin 9
LL_GPIO_AF_EVENTOUT_PIN_10	EVENTOUT on pin 10
LL_GPIO_AF_EVENTOUT_PIN_11	EVENTOUT on pin 11
LL_GPIO_AF_EVENTOUT_PIN_12	EVENTOUT on pin 12
LL_GPIO_AF_EVENTOUT_PIN_13	EVENTOUT on pin 13
LL_GPIO_AF_EVENTOUT_PIN_14	EVENTOUT on pin 14
LL_GPIO_AF_EVENTOUT_PIN_15	EVENTOUT on pin 15

EVENTOUT Port

LL_GPIO_AF_EVENTOUT_PORT_A	EVENTOUT on port A
LL_GPIO_AF_EVENTOUT_PORT_B	EVENTOUT on port B
LL_GPIO_AF_EVENTOUT_PORT_C	EVENTOUT on port C
LL_GPIO_AF_EVENTOUT_PORT_D	EVENTOUT on port D
LL_GPIO_AF_EVENTOUT_PORT_E	EVENTOUT on port E

GPIO Exported Constants

LL_GPIO_SPEED_FREQ_LOW	Select I/O low output speed
LL_GPIO_SPEED_FREQ_MEDIUM	Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH	Select I/O high output speed

54 LL I2C Generic Driver

54.1 I2C Firmware driver registers structures

54.1.1 LL_I2C_InitTypeDef

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t ClockSpeed***
- ***uint32_t DutyCycle***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***
  Specifies the peripheral mode. This parameter can be a value of ***I2C_LL_EC_PERIPHERAL_MODE***This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::ClockSpeed***
  Specifies the clock frequency. This parameter must be set to a value lower than 400kHz (in Hz)This feature can be modified afterwards using unitary function **LL_I2C_SetClockPeriod()** or **LL_I2C_SetDutyCycle()** or **LL_I2C_SetClockSpeedMode()** or **LL_I2C_ConfigSpeed()**.
- ***uint32_t LL_I2C_InitTypeDef::DutyCycle***
  Specifies the I2C fast mode duty cycle. This parameter can be a value of ***I2C_LL_EC_DUTYCYCLE***This feature can be modified afterwards using unitary function **LL_I2C_SetDutyCycle()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***
  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FFThis feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***
  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of ***I2C_LL_EC_I2C_ACKNOWLEDGE***This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***
  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of ***I2C_LL_EC_OWNADDRESS1***This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

54.2 I2C Firmware driver API description

54.2.1 Detailed description of functions

LL_I2C_Enable

Function name	**__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)**
Function description	Enable I2C peripheral (PE = 1).

Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name	**__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)**
Function description	Disable I2C peripheral (PE = 0).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabled (I2C_TypeDef * I2Cx)**
Function description	Check if the I2C peripheral is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 PE LL_I2C_IsEnabled

LL_I2C_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Enable DMA transmission requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 DMAEN LL_I2C_EnableDMAReq_TX

LL_I2C_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Disable DMA transmission requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR2 DMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (I2C_TypeDef * I2Cx)**
Function description	Check if DMA transmission requests are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 DMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Enable DMA reception requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 DMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Disable DMA reception requests.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 DMAEN LL_I2C_DisableDMAReq_RX

LL_I2C_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (I2C_TypeDef * I2Cx)**
Function description	Check if DMA reception requests are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR2 DMAEN LL_I2C_IsEnabledDMAReq_RX

reference:

LL_I2C_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• DR DR LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name	**__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)**
Function description	Enable Clock stretching.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name	**__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)**
Function description	Disable Clock stretching.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (I2C_TypeDef * I2Cx)**
Function description	Check if Clock stretching is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableGeneralCall

Function name	**__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)**
Function description	Enable General Call.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• When enabled the Address 0x00 is ACKed.
Reference Manual to LL API cross reference:	• CR1 ENGC LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name	**__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)**
Function description	Disable General Call.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• When disabled the Address 0x00 is NACKed.
Reference Manual to LL API cross reference:	• CR1 ENGC LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (I2C_TypeDef * I2Cx)**
Function description	Check if General Call is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 ENGC LL_I2C_IsEnabledGeneralCall

LL_I2C_SetOwnAddress1

Function name	**__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)**

Function description	Set the Own Address1.
Parameters	• **I2Cx:** I2C Instance. • **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF. • **OwnAddrSize:** This parameter can be one of the following values: – LL_I2C_OWNADDRESS1_7BIT – LL_I2C_OWNADDRESS1_10BIT
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR1 ADD0 LL_I2C_SetOwnAddress1 • OAR1 ADD1_7 LL_I2C_SetOwnAddress1 • OAR1 ADD8_9 LL_I2C_SetOwnAddress1 • OAR1 ADDMODE LL_I2C_SetOwnAddress1

LL_I2C_SetOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2)**
Function description	Set the 7bits Own Address2.
Parameters	• **I2Cx:** I2C Instance. • **OwnAddress2:** This parameter must be a value between Min_Data=0 and Max_Data=0x7F.
Return values	• **None:**
Notes	• This action has no effect if own address2 is enabled.
Reference Manual to LL API cross reference:	• OAR2 ADD2 LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Enable acknowledge on Own Address2 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• OAR2 ENDUAL LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name	**__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Disable acknowledge on Own Address2 match address.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**

Reference Manual to LL API cross reference:	• OAR2 ENDUAL LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (I2C_TypeDef * I2Cx)**
Function description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• OAR2 ENDUAL LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetPeriphClock

Function name	**__STATIC_INLINE void LL_I2C_SetPeriphClock (I2C_TypeDef * I2Cx, uint32_t PeriphClock)**
Function description	Configure the Peripheral clock frequency.
Parameters	• **I2Cx:** I2C Instance. • **PeriphClock:** Peripheral Clock (in Hz)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 FREQ LL_I2C_SetPeriphClock

LL_I2C_GetPeriphClock

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetPeriphClock (I2C_TypeDef * I2Cx)**
Function description	Get the Peripheral clock frequency.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** of Peripheral Clock (in Hz)
Reference Manual to LL API cross reference:	• CR2 FREQ LL_I2C_GetPeriphClock

LL_I2C_SetDutyCycle

Function name	**__STATIC_INLINE void LL_I2C_SetDutyCycle (I2C_TypeDef * I2Cx, uint32_t DutyCycle)**
Function description	Configure the Duty cycle (Fast mode only).
Parameters	• **I2Cx:** I2C Instance. • **DutyCycle:** This parameter can be one of the following values: – LL_I2C_DUTYCYCLE_2

– LL_I2C_DUTYCYCLE_16_9

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR DUTY LL_I2C_SetDutyCycle

LL_I2C_GetDutyCycle

Function name
__STATIC_INLINE uint32_t LL_I2C_GetDutyCycle (I2C_TypeDef * I2Cx)

Function description
Get the Duty cycle (Fast mode only).

Parameters
- **I2Cx:** I2C Instance.

Return values
- **Returned:** value can be one of the following values:
  – LL_I2C_DUTYCYCLE_2
  – LL_I2C_DUTYCYCLE_16_9

Reference Manual to LL API cross reference:
- CCR DUTY LL_I2C_GetDutyCycle

LL_I2C_SetClockSpeedMode

Function name
__STATIC_INLINE void LL_I2C_SetClockSpeedMode (I2C_TypeDef * I2Cx, uint32_t ClockSpeedMode)

Function description
Configure the I2C master clock speed mode.

Parameters
- **I2Cx:** I2C Instance.
- **ClockSpeedMode:** This parameter can be one of the following values:
  – LL_I2C_CLOCK_SPEED_STANDARD_MODE
  – LL_I2C_CLOCK_SPEED_FAST_MODE

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCR FS LL_I2C_SetClockSpeedMode

LL_I2C_GetClockSpeedMode

Function name
__STATIC_INLINE uint32_t LL_I2C_GetClockSpeedMode (I2C_TypeDef * I2Cx)

Function description
Get the the I2C master speed mode.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **Returned:** value can be one of the following values:
  – LL_I2C_CLOCK_SPEED_STANDARD_MODE
  – LL_I2C_CLOCK_SPEED_FAST_MODE

Reference Manual to LL API cross reference:
- CCR FS LL_I2C_GetClockSpeedMode

LL_I2C_SetRiseTime

Function name	**__STATIC_INLINE void LL_I2C_SetRiseTime (I2C_TypeDef * I2Cx, uint32_t RiseTime)**
Function description	Configure the SCL, SDA rising time.
Parameters	• **I2Cx:** I2C Instance. • **RiseTime:** This parameter must be a value between Min_Data=0x02 and Max_Data=0x3F.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• TRISE TRISE LL_I2C_SetRiseTime

LL_I2C_GetRiseTime

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetRiseTime (I2C_TypeDef * I2Cx)**
Function description	Get the SCL, SDA rising time.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x02 and Max_Data=0x3F
Reference Manual to LL API cross reference:	• TRISE TRISE LL_I2C_GetRiseTime

LL_I2C_SetClockPeriod

Function name	**__STATIC_INLINE void LL_I2C_SetClockPeriod (I2C_TypeDef * I2Cx, uint32_t ClockPeriod)**
Function description	Configure the SCL high and low period.
Parameters	• **I2Cx:** I2C Instance. • **ClockPeriod:** This parameter must be a value between Min_Data=0x004 and Max_Data=0xFFF, except in FAST DUTY mode where Min_Data=0x001.
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CCR CCR LL_I2C_SetClockPeriod

LL_I2C_GetClockPeriod

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetClockPeriod (I2C_TypeDef * I2Cx)**

Function description	Get the SCL high and low period.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x004 and Max_Data=0xFFF, except in FAST DUTY mode where Min_Data=0x001.
Reference Manual to LL API cross reference:	• CCR CCR LL_I2C_GetClockPeriod

LL_I2C_ConfigSpeed

Function name	**__STATIC_INLINE void LL_I2C_ConfigSpeed (I2C_TypeDef * I2Cx, uint32_t PeriphClock, uint32_t ClockSpeed, uint32_t DutyCycle)**
Function description	Configure the SCL speed.
Parameters	• **I2Cx:** I2C Instance. • **PeriphClock:** Peripheral Clock (in Hz) • **ClockSpeed:** This parameter must be a value lower than 400kHz (in Hz). • **DutyCycle:** This parameter can be one of the following values: – LL_I2C_DUTYCYCLE_2 – LL_I2C_DUTYCYCLE_16_9
Return values	• **None:**
Notes	• This bit can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR2 FREQ LL_I2C_ConfigSpeed • TRISE TRISE LL_I2C_ConfigSpeed • CCR FS LL_I2C_ConfigSpeed • CCR DUTY LL_I2C_ConfigSpeed • CCR CCR LL_I2C_ConfigSpeed

LL_I2C_SetMode

Function name	**__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)**
Function description	Configure peripheral mode.
Parameters	• **I2Cx:** I2C Instance. • **PeripheralMode:** This parameter can be one of the following values: – LL_I2C_MODE_I2C – LL_I2C_MODE_SMBUS_HOST – LL_I2C_MODE_SMBUS_DEVICE – LL_I2C_MODE_SMBUS_DEVICE_ARP
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:	• CR1 SMBUS LL_I2C_SetMode • CR1 SMBTYPE LL_I2C_SetMode • CR1 ENARP LL_I2C_SetMode

LL_I2C_GetMode

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)**
Function description	Get peripheral mode.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_MODE_I2C – LL_I2C_MODE_SMBUS_HOST – LL_I2C_MODE_SMBUS_DEVICE – LL_I2C_MODE_SMBUS_DEVICE_ARP
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 SMBUS LL_I2C_GetMode • CR1 SMBTYPE LL_I2C_GetMode • CR1 ENARP LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name	**__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Enable SMBus alert (Host or Device mode)
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.
Reference Manual to LL API cross reference:	• CR1 ALERT LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name	**__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)**
Function description	Disable SMBus alert (Host or Device mode)
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:
- CR1 ALERT LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name **__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (I2C_TypeDef * I2Cx)**

Function description Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **State:** of bit (1 or 0).

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:
- CR1 ALERT LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name **__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)**

Function description Enable SMBus Packet Error Calculation (PEC).

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:
- CR1 ENPEC LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name **__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)**

Function description Disable SMBus Packet Error Calculation (PEC).

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 ENPEC LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• CR1 ENPEC LL_I2C_IsEnabledSMBusPEC

LL_I2C_EnableIT_TX

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)**
Function description	Enable TXE interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_EnableIT_TX • CR2 ITBUFEN LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)**
Function description	Disable TXE interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_DisableIT_TX • CR2 ITBUFEN LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (I2C_TypeDef * I2Cx)**
Function description	Check if the TXE Interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_IsEnabledIT_TX • CR2 ITBUFEN LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)**
Function description	Enable RXNE interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_EnableIT_RX • CR2 ITBUFEN LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)**
Function description	Disable RXNE interrupt.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_DisableIT_RX • CR2 ITBUFEN LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (I2C_TypeDef * I2Cx)**
Function description	Check if the RXNE Interrupt is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_IsEnabledIT_RX • CR2 ITBUFEN LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_EVT

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_EVT (I2C_TypeDef * I2Cx)**
Function description	Enable Events interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Any of these events will generate interrupt : Start Bit (SB) Address sent, Address matched (ADDR) 10-bit header sent (ADD10) Stop detection (STOPF) Byte transfer finished (BTF) • Any of these events will generate interrupt if Buffer interrupts are enabled too(using unitary function LL_I2C_EnableIT_BUF()) : Receive buffer not empty (RXNE) Transmit buffer empty (TXE)
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_EnableIT_EVT

LL_I2C_DisableIT_EVT

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_EVT (I2C_TypeDef * I2Cx)**
Function description	Disable Events interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Any of these events will generate interrupt : Start Bit (SB) Address sent, Address matched (ADDR) 10-bit header sent (ADD10) Stop detection (STOPF) Byte transfer finished (BTF) Receive buffer not empty (RXNE) Transmit buffer empty (TXE)
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_DisableIT_EVT

LL_I2C_IsEnabledIT_EVT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_EVT (I2C_TypeDef * I2Cx)**
Function description	Check if Events interrupts are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ITEVTEN LL_I2C_IsEnabledIT_EVT

LL_I2C_EnableIT_BUF

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_BUF (I2C_TypeDef * I2Cx)**
Function description	Enable Buffer interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Any of these Buffer events will generate interrupt if Events interrupts are enabled too(using unitary function LL_I2C_EnableIT_EVT()) : Receive buffer not empty (RXNE) Transmit buffer empty (TXE)
Reference Manual to LL API cross reference:	• CR2 ITBUFEN LL_I2C_EnableIT_BUF

LL_I2C_DisableIT_BUF

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_BUF (I2C_TypeDef * I2Cx)**
Function description	Disable Buffer interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Any of these Buffer events will generate interrupt : Receive buffer not empty (RXNE) Transmit buffer empty (TXE)
Reference Manual to LL API cross reference:	• CR2 ITBUFEN LL_I2C_DisableIT_BUF

LL_I2C_IsEnabledIT_BUF

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_BUF (I2C_TypeDef * I2Cx)**
Function description	Check if Buffer interrupts are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ITBUFEN LL_I2C_IsEnabledIT_BUF

LL_I2C_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Enable Error interrupts.
Parameters	• **I2Cx:** I2C Instance.

Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • Any of these errors will generate interrupt : Bus Error detection (BERR) Arbitration Loss (ARLO) Acknowledge Failure(AF) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (SMBALERT)
Reference Manual to LL API cross reference:	• CR2 ITERREN LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Disable Error interrupts.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • Any of these errors will generate interrupt : Bus Error detection (BERR) Arbitration Loss (ARLO) Acknowledge Failure(AF) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (SMBALERT)
Reference Manual to LL API cross reference:	• CR2 ITERREN LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (I2C_TypeDef * I2Cx)**
Function description	Check if Error interrupts are enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ITERREN LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Transmit data register empty flag.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.
Reference Manual to LL API cross reference:	• SR1 TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_BTF

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BTF (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Byte Transfer Finished flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR1 BTF LL_I2C_IsActiveFlag_BTF

LL_I2C_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Receive data register not empty flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.
Reference Manual to LL API cross reference:	• SR1 RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_SB

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_SB (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Start Bit (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When No Start condition. SET: When Start condition is generated.
Reference Manual to LL API cross reference:	• SR1 SB LL_I2C_IsActiveFlag_SB

LL_I2C_IsActiveFlag_ADDR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Address sent (master mode) or Address matched flag (slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When the address is fully sent (master mode) or when the received slave address matched with one of the enabled slave address (slave mode).
Reference Manual to LL API cross reference:	• SR1 ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_ADD10

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADD10 (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of 10-bit header sent (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: When no ADD10 event occured. SET: When the master has sent the first address byte (header).
Reference Manual to LL API cross reference:	• SR1 ADD10 LL_I2C_IsActiveFlag_ADD10

LL_I2C_IsActiveFlag_AF

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_AF (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Acknowledge failure flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: No acknowledge failure. SET: When an acknowledge failure is received after a byte transmission.
Reference Manual to LL API cross reference:	• SR1 AF LL_I2C_IsActiveFlag_AF

LL_I2C_IsActiveFlag_STOP

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Stop detection flag (slave mode).

Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a Stop condition is detected.
Reference Manual to LL API cross reference:	• SR1 STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_BERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Bus error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.
Reference Manual to LL API cross reference:	• SR1 BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Arbitration lost flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When arbitration lost.
Reference Manual to LL API cross reference:	• SR1 ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Overrun/Underrun flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).
Reference Manual to LL API cross	• SR1 OVR LL_I2C_IsActiveFlag_OVR

reference:

LL_I2C_IsActiveSMBusFlag_PECERR

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus PEC error flag in reception.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• SR1 PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus Timeout detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• SR1 TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus alert flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• SR1 SMBALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Bus Busy flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Clear default value. SET: When a Start condition is detected.
Reference Manual to LL API cross reference:	• SR2 BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_IsActiveFlag_DUAL

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_DUAL (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Dual flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Received address matched with OAR1. SET: Received address matched with OAR2.
Reference Manual to LL API cross reference:	• SR2 DUALF LL_I2C_IsActiveFlag_DUAL

LL_I2C_IsActiveSMBusFlag_SMBHOST

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_SMBHOST (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus Host address reception (Slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: No SMBus Host address SET: SMBus Host address received. • This status is cleared by hardware after a STOP condition or repeated START condition.
Reference Manual to LL API cross reference:	• SR2 SMBHOST LL_I2C_IsActiveSMBusFlag_SMBHOST

LL_I2C_IsActiveSMBusFlag_SMBDEFAULT

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_SMBDEFAULT (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of SMBus Device default address reception (Slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance. • RESET: No SMBus Device default address SET: SMBus Device default address received. • This status is cleared by hardware after a STOP condition or repeated START condition.
Reference Manual to LL API cross reference:	• SR2 SMBDEFAULT LL_I2C_IsActiveSMBusFlag_SMBDEFAULT

LL_I2C_IsActiveFlag_GENCALL

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_GENCALL (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of General call address reception (Slave mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: No Generall call address SET: General call address received. • This status is cleared by hardware after a STOP condition or repeated START condition.
Reference Manual to LL API cross reference:	• SR2 GENCALL LL_I2C_IsActiveFlag_GENCALL

LL_I2C_IsActiveFlag_MSL

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_MSL (I2C_TypeDef * I2Cx)**
Function description	Indicate the status of Master/Slave flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Notes	• RESET: Slave Mode. SET: Master Mode.
Reference Manual to LL API cross reference:	• SR2 MSL LL_I2C_IsActiveFlag_MSL

LL_I2C_ClearFlag_ADDR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)**
Function description	Clear Address Matched flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the I2Cx_SR1 register followed by a read access to the I2Cx_SR2 register.
Reference Manual to LL API cross reference:	• SR1 ADDR LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_AF

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_AF (I2C_TypeDef * I2Cx)**
Function description	Clear Acknowledge failure flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR1 AF LL_I2C_ClearFlag_AF

LL_I2C_ClearFlag_STOP

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)**
Function description	Clear Stop detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the I2Cx_SR1 register followed by a write access to I2Cx_CR1 register.
Reference Manual to LL API cross reference:	• SR1 STOPF LL_I2C_ClearFlag_STOP • CR1 PE LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_BERR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)**
Function description	Clear Bus error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to	• SR1 BERR LL_I2C_ClearFlag_BERR

LL API cross reference:

LL_I2C_ClearFlag_ARLO

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)**
Function description	Clear Arbitration lost flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR1 ARLO LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)**
Function description	Clear Overrun/Underrun flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR1 OVR LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name	**__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)**
Function description	Clear SMBus PEC error flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR1 PECERR LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name	**__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)**
Function description	Clear SMBus Timeout detection flag.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx

Instance.

Reference Manual to LL API cross reference:
- SR1 TIMEOUT LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name **__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)**

Function description Clear SMBus Alert flag.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:
- SR1 SMBALERT LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableReset

Function name **__STATIC_INLINE void LL_I2C_EnableReset (I2C_TypeDef * I2Cx)**

Function description Enable Reset of I2C peripheral.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR1 SWRST LL_I2C_EnableReset

LL_I2C_DisableReset

Function name **__STATIC_INLINE void LL_I2C_DisableReset (I2C_TypeDef * I2Cx)**

Function description Disable Reset of I2C peripheral.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR1 SWRST LL_I2C_DisableReset

LL_I2C_IsResetEnabled

Function name **__STATIC_INLINE uint32_t LL_I2C_IsResetEnabled (I2C_TypeDef * I2Cx)**

Function description Check if the I2C peripheral is under reset state or not.

Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 SWRST LL_I2C_IsResetEnabled

LL_I2C_AcknowledgeNextData

Function name	**__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)**
Function description	Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.
Parameters	• **I2Cx:** I2C Instance. • **TypeAcknowledge:** This parameter can be one of the following values: – LL_I2C_ACK – LL_I2C_NACK
Return values	• **None:**
Notes	• Usage in Slave or Master mode.
Reference Manual to LL API cross reference:	• CR1 ACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name	**__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)**
Function description	Generate a START or RESTART condition.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.
Reference Manual to LL API cross reference:	• CR1 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name	**__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)**
Function description	Generate a STOP condition after the current byte transfer (master mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR1 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableBitPOS

Function name	**__STATIC_INLINE void LL_I2C_EnableBitPOS (I2C_TypeDef * I2Cx)**
Function description	Enable bit POS (master/host mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• In that case, the ACK bit controls the (N)ACK of the next byte received or the PEC bit indicates that the next byte in shift register is a PEC.
Reference Manual to LL API cross reference:	• CR1 POS LL_I2C_EnableBitPOS

LL_I2C_DisableBitPOS

Function name	**__STATIC_INLINE void LL_I2C_DisableBitPOS (I2C_TypeDef * I2Cx)**
Function description	Disable bit POS (master/host mode).
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• In that case, the ACK bit controls the (N)ACK of the current byte received or the PEC bit indicates that the current byte in shift register is a PEC.
Reference Manual to LL API cross reference:	• CR1 POS LL_I2C_DisableBitPOS

LL_I2C_IsEnabledBitPOS

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledBitPOS (I2C_TypeDef * I2Cx)**
Function description	Check if bit POS is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 POS LL_I2C_IsEnabledBitPOS

LL_I2C_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection**

(I2C_TypeDef * I2Cx)

Function description	Indicate the value of transfer direction.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Returned:** value can be one of the following values: – LL_I2C_DIRECTION_WRITE – LL_I2C_DIRECTION_READ
Notes	• RESET: Bus is in read transfer (peripheral point of view). SET: Bus is in write transfer (peripheral point of view).
Reference Manual to LL API cross reference:	• SR2 TRA LL_I2C_GetTransferDirection

LL_I2C_EnableLastDMA

Function name	**__STATIC_INLINE void LL_I2C_EnableLastDMA (I2C_TypeDef * I2Cx)**
Function description	Enable DMA last transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This action mean that next DMA EOT is the last transfer.
Reference Manual to LL API cross reference:	• CR2 LAST LL_I2C_EnableLastDMA

LL_I2C_DisableLastDMA

Function name	**__STATIC_INLINE void LL_I2C_DisableLastDMA (I2C_TypeDef * I2Cx)**
Function description	Disable DMA last transfer.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **None:**
Notes	• This action mean that next DMA EOT is not the last transfer.
Reference Manual to LL API cross reference:	• CR2 LAST LL_I2C_DisableLastDMA

LL_I2C_IsEnabledLastDMA

Function name	**__STATIC_INLINE uint32_t LL_I2C_IsEnabledLastDMA (I2C_TypeDef * I2Cx)**
Function description	Check if DMA last transfer is enabled or disabled.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **State:** of bit (1 or 0).
Reference Manual to	• CR2 LAST LL_I2C_IsEnabledLastDMA

LL API cross reference:

LL_I2C_EnableSMBusPECCompare

Function name **__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)**

Function description Enable transfer or internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This feature is cleared by hardware when the PEC byte is transferred or compared, or by a START or STOP condition, it is also cleared by software.

Reference Manual to LL API cross reference:
- CR1 PEC LL_I2C_EnableSMBusPECCompare

LL_I2C_DisableSMBusPECCompare

Function name **__STATIC_INLINE void LL_I2C_DisableSMBusPECCompare (I2C_TypeDef * I2Cx)**

Function description Disable transfer or internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters
- **I2Cx:** I2C Instance.

Return values
- **None:**

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:
- CR1 PEC LL_I2C_DisableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name **__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (I2C_TypeDef * I2Cx)**

Function description Check if the SMBus Packet Error byte transfer or internal comparison is requested or not.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **State:** of bit (1 or 0).

Notes
- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:	• CR1 PEC LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name	**__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)**
Function description	Get the SMBus Packet Error byte calculated.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Notes	• Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
Reference Manual to LL API cross reference:	• SR2 PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)**
Function description	Read Receive Data register.
Parameters	• **I2Cx:** I2C Instance.
Return values	• **Value:** between Min_Data=0x0 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• DR DR LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name	**__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)**
Function description	Write in Transmit Data Register .
Parameters	• **I2Cx:** I2C Instance. • **Data:** Value between Min_Data=0x0 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_I2C_TransmitData8

LL_I2C_Init

Function name	**uint32_t LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)**
Function description	Initialize the I2C registers according to the specified parameters in

I2C_InitStruct.

Parameters
- **I2Cx:** I2C Instance.
- **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: I2C registers are initialized
  – ERROR: Not applicable

LL_I2C_DeInit

Function name **uint32_t LL_I2C_DeInit (I2C_TypeDef * I2Cx)**

Function description De-initialize the I2C registers to their default reset values.

Parameters
- **I2Cx:** I2C Instance.

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: I2C registers are de-initialized
  – ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name **void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)**

Function description Set each LL_I2C_InitTypeDef field to default value.

Parameters
- **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.

Return values
- **None:**

54.3 I2C Firmware driver defines

54.3.1 I2C

Master Clock Speed Mode

LL_I2C_CLOCK_SPEED_STANDARD_MODE	Master clock speed range is standard mode
LL_I2C_CLOCK_SPEED_FAST_MODE	Master clock speed range is fast mode

Read Write Direction

LL_I2C_DIRECTION_WRITE	Bus is in write transfer
LL_I2C_DIRECTION_READ	Bus is in read transfer

Fast Mode Duty Cycle

LL_I2C_DUTYCYCLE_2	I2C fast mode Tlow/Thigh = 2
LL_I2C_DUTYCYCLE_16_9	I2C fast mode Tlow/Thigh = 16/9

Get Flags Defines

LL_I2C_SR1_SB	Start Bit (master mode)
LL_I2C_SR1_ADDR	Address sent (master mode) or Address matched flag (slave mode)
LL_I2C_SR1_BTF	Byte Transfer Finished flag

LL_I2C_SR1_ADD10	10-bit header sent (master mode)
LL_I2C_SR1_STOPF	Stop detection flag (slave mode)
LL_I2C_SR1_RXNE	Data register not empty (receivers)
LL_I2C_SR1_TXE	Data register empty (transmitters)
LL_I2C_SR1_BERR	Bus error
LL_I2C_SR1_ARLO	Arbitration lost
LL_I2C_SR1_AF	Acknowledge failure flag
LL_I2C_SR1_OVR	Overrun/Underrun
LL_I2C_SR1_PECERR	PEC Error in reception (SMBus mode)
LL_I2C_SR1_TIMEOUT	Timeout detection flag (SMBus mode)
LL_I2C_SR1_SMALERT	SMBus alert (SMBus mode)
LL_I2C_SR2_MSL	Master/Slave flag
LL_I2C_SR2_BUSY	Bus busy flag
LL_I2C_SR2_TRA	Transmitter/receiver direction
LL_I2C_SR2_GENCALL	General call address (Slave mode)
LL_I2C_SR2_SMBDEFAULT	SMBus Device default address (Slave mode)
LL_I2C_SR2_SMBHOST	SMBus Host address (Slave mode)
LL_I2C_SR2_DUALF	Dual flag (Slave mode)

Acknowledge Generation

LL_I2C_ACK	ACK is sent after current received byte.
LL_I2C_NACK	NACK is sent after current received byte.

IT Defines

LL_I2C_CR2_ITEVTEN	Events interrupts enable
LL_I2C_CR2_ITBUFEN	Buffer interrupts enable
LL_I2C_CR2_ITERREN	Error interrupts enable

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT	Own address 1 is a 7-bit address.
LL_I2C_OWNADDRESS1_10BIT	Own address 1 is a 10-bit address.

Peripheral Mode

LL_I2C_MODE_I2C	I2C Master or Slave mode
LL_I2C_MODE_SMBUS_HOST	SMBus Host address acknowledge
LL_I2C_MODE_SMBUS_DEVICE	SMBus Device default mode (Default address not acknowledge)
LL_I2C_MODE_SMBUS_DEVICE_ARP	SMBus Device Default address acknowledge

Exported_Macros_Helper

__LL_I2C_FREQ_HZ_TO_MHZ

Description:

- Convert Peripheral Clock Frequency in

Mhz.

Parameters:

- __PCLK__: This parameter must be a value of peripheral clock (in Hz).

Return value:

- Value: of peripheral clock (in Mhz)

__LL_I2C_FREQ_MHZ_TO_HZ

Description:

- Convert Peripheral Clock Frequency in Hz.

Parameters:

- __PCLK__: This parameter must be a value of peripheral clock (in Mhz).

Return value:

- Value: of peripheral clock (in Hz)

__LL_I2C_RISE_TIME

Description:

- Compute I2C Clock rising time.

Parameters:

- __FREQRANGE__: This parameter must be a value of peripheral clock (in Mhz).
- __SPEED__: This parameter must be a value lower than 400kHz (in Hz).

Return value:

- Value: between Min_Data=0x02 and Max_Data=0x3F

__LL_I2C_SPEED_TO_CCR

Description:

- Compute Speed clock range to a Clock Control Register (I2C_CCR_CCR) value.

Parameters:

- __PCLK__: This parameter must be a value of peripheral clock (in Hz).
- __SPEED__: This parameter must be a value lower than 400kHz (in Hz).
- __DUTYCYCLE__: This parameter can be one of the following values:
  - LL_I2C_DUTYCYCLE_2
  - LL_I2C_DUTYCYCLE_16_9

Return value:

- Value: between Min_Data=0x004 and Max_Data=0xFFF, except in FAST DUTY mode where Min_Data=0x001.

__LL_I2C_SPEED_STANDARD_TO_CCR

Description:

- Compute Speed Standard clock range to a Clock Control Register (I2C_CCR_CCR) value.

Parameters:

- __PCLK__: This parameter must be a value of peripheral clock (in Hz).
- __SPEED__: This parameter must be a value lower than 100kHz (in Hz).

Return value:

- Value: between Min_Data=0x004 and Max_Data=0xFFF.

__LL_I2C_SPEED_FAST_TO_CCR

Description:

- Compute Speed Fast clock range to a Clock Control Register (I2C_CCR_CCR) value.

Parameters:

- __PCLK__: This parameter must be a value of peripheral clock (in Hz).
- __SPEED__: This parameter must be a value between Min_Data=100Khz and Max_Data=400Khz (in Hz).
- __DUTYCYCLE__: This parameter can be one of the following values:
  - LL_I2C_DUTYCYCLE_2
  - LL_I2C_DUTYCYCLE_16_9

Return value:

- Value: between Min_Data=0x001 and Max_Data=0xFFF

__LL_I2C_10BIT_ADDRESS

Description:

- Get the Least significant bits of a 10-Bits address.

Parameters:

- __ADDRESS__: This parameter must be a value of a 10-Bits slave address.

Return value:

- Value: between Min_Data=0x00 and Max_Data=0xFF

__LL_I2C_10BIT_HEADER_WRITE

Description:

- Convert a 10-Bits address to a 10-Bits header with Write direction.

Parameters:

- __ADDRESS__: This parameter must be a value of a 10-Bits slave address.

Return value:

- Value: between Min_Data=0xF0 and Max_Data=0xF6

__LL_I2C_10BIT_HEADER_READ

Description:

- Convert a 10-Bits address to a 10-Bits header with Read direction.

Parameters:

- __ADDRESS__: This parameter must be a value of a 10-Bits slave address.

Return value:

- Value: between Min_Data=0xF1 and Max_Data=0xF7

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

55 LL I2S Generic Driver

55.1 I2S Firmware driver registers structures

55.1.1 LL_I2S_InitTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t ClockPolarity***

Field Documentation

- ***uint32_t LL_I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_LL_EC_MODE***This feature can be modified afterwards using unitary function **LL_I2S_SetTransferMode()**.
- ***uint32_t LL_I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_LL_EC_STANDARD***This feature can be modified afterwards using unitary function **LL_I2S_SetStandard()**.
- ***uint32_t LL_I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_LL_EC_DATA_FORMAT***This feature can be modified afterwards using unitary function **LL_I2S_SetDataFormat()**.
- ***uint32_t LL_I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_LL_EC_MCLK_OUTPUT***This feature can be modified afterwards using unitary functions **LL_I2S_EnableMasterClock()** or **LL_I2S_DisableMasterClock**.
- ***uint32_t LL_I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_LL_EC_AUDIO_FREQ***Audio Frequency can be modified afterwards using Reference manual formulas to calculate Prescaler Linear, Parity and unitary functions **LL_I2S_SetPrescalerLinear()** and **LL_I2S_SetPrescalerParity()** to set it.
- ***uint32_t LL_I2S_InitTypeDef::ClockPolarity***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_LL_EC_POLARITY***This feature can be modified afterwards using unitary function **LL_I2S_SetClockPolarity()**.

55.2 I2S Firmware driver API description

55.2.1 Detailed description of functions

LL_I2S_Enable

Function name	**__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)**
Function description	Select I2S mode and Enable I2S peripheral.
Parameters	• **SPIx:** SPI Instance

Return values
- **None:**

Reference Manual to LL API cross reference:
- I2SCFGR I2SMOD LL_I2S_Enable
- I2SCFGR I2SE LL_I2S_Enable

LL_I2S_Disable

Function name **__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)**

Function description Disable I2S peripheral.

Parameters
- **SPIx:** SPI Instance

Return values
- **None:**

Reference Manual to LL API cross reference:
- I2SCFGR I2SE LL_I2S_Disable

LL_I2S_IsEnabled

Function name **__STATIC_INLINE uint32_t LL_I2S_IsEnabled (SPI_TypeDef * SPIx)**

Function description Check if I2S peripheral is enabled.

Parameters
- **SPIx:** SPI Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- I2SCFGR I2SE LL_I2S_IsEnabled

LL_I2S_SetDataFormat

Function name **__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataFormat)**

Function description Set I2S data frame length.

Parameters
- **SPIx:** SPI Instance
- **DataFormat:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Return values
- **None:**

Reference Manual to LL API cross reference:
- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name **__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (SPI_TypeDef * SPIx)**

Function description	Get I2S data frame length.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_DATAFORMAT_16B – LL_I2S_DATAFORMAT_16B_EXTENDED – LL_I2S_DATAFORMAT_24B – LL_I2S_DATAFORMAT_32B
Reference Manual to LL API cross reference:	• I2SCFGR DATLEN LL_I2S_GetDataFormat • I2SCFGR CHLEN LL_I2S_GetDataFormat

LL_I2S_SetClockPolarity

Function name	**__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)**
Function description	Set I2S clock polarity.
Parameters	• **SPIx:** SPI Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_I2S_POLARITY_LOW – LL_I2S_POLARITY_HIGH
Return values	• **None:**
Reference Manual to LL API cross reference:	• I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (SPI_TypeDef * SPIx)**
Function description	Get I2S clock polarity.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_POLARITY_LOW – LL_I2S_POLARITY_HIGH
Reference Manual to LL API cross reference:	• I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name	**__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)**
Function description	Set I2S standard protocol.
Parameters	• **SPIx:** SPI Instance • **Standard:** This parameter can be one of the following values:

- LL_I2S_STANDARD_PHILIPS
- LL_I2S_STANDARD_MSB
- LL_I2S_STANDARD_LSB
- LL_I2S_STANDARD_PCM_SHORT
- LL_I2S_STANDARD_PCM_LONG

Return values
- **None:**

Reference Manual to LL API cross reference:
- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name **__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)**

Function description Get I2S standard protocol.

Parameters
- **SPIx:** SPI Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:
- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name **__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)**

Function description Set I2S transfer mode.

Parameters
- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Return values
- **None:**

Reference Manual to LL API cross reference:
- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name **__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (SPI_TypeDef * SPIx)**

Function description Get I2S transfer mode.

Parameters
- **SPIx:** SPI Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Reference Manual to LL API cross reference:
- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_SetPrescalerLinear

Function name **__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint8_t PrescalerLinear)**

Function description Set I2S linear prescaler.

Parameters
- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Return values
- **None:**

Reference Manual to LL API cross reference:
- I2SPR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name **__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (SPI_TypeDef * SPIx)**

Function description Get I2S linear prescaler.

Parameters
- **SPIx:** SPI Instance

Return values
- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Reference Manual to LL API cross reference:
- I2SPR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name **__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)**

Function description Set I2S parity prescaler.

Parameters
- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values
- **None:**

Reference Manual to LL API cross reference:	• I2SPR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name	**__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (SPI_TypeDef * SPIx)**
Function description	Get I2S parity prescaler.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_I2S_PRESCALER_PARITY_EVEN – LL_I2S_PRESCALER_PARITY_ODD
Reference Manual to LL API cross reference:	• I2SPR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name	**__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)**
Function description	Enable the master clock ouput (Pin MCK)
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• I2SPR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name	**__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)**
Function description	Disable the master clock ouput (Pin MCK)
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• I2SPR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (SPI_TypeDef * SPIx)**
Function description	Check if the master clock ouput (Pin MCK) is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• I2SPR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if Rx buffer is not empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR RXNE LL_I2S_IsActiveFlag_RXNE

LL_I2S_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer is empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TXE LL_I2S_IsActiveFlag_TXE

LL_I2S_IsActiveFlag_BSY

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY (SPI_TypeDef * SPIx)**
Function description	Get busy flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR BSY LL_I2S_IsActiveFlag_BSY

LL_I2S_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Get overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• SR OVR LL_I2S_IsActiveFlag_OVR

reference:

LL_I2S_IsActiveFlag_UDR

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (SPI_TypeDef * SPIx)**
Function description	Get underrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_FRE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Get frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR FRE LL_I2S_IsActiveFlag_FRE

LL_I2S_IsActiveFlag_CHSIDE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_CHSIDE (SPI_TypeDef * SPIx)**
Function description	Get channel side flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• 0: Channel Left has to be transmitted or has been received 1: Channel Right has to be transmitted or has been received It has no significance in PCM mode.
Reference Manual to LL API cross reference:	• SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE

LL_I2S_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Clear overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• SR OVR LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_UDR

Function name	**__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)**
Function description	Clear underrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR UDR LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_FRE

Function name	**__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)**
Function description	Clear frame format error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR FRE LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Enable error IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_EnableIT_ERR

LL_I2S_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Enable Rx buffer not empty IT.
Parameters	• **SPIx:** SPI Instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_EnableIT_RXNE

LL_I2S_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_I2S_EnableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Enable Tx buffer empty IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_I2S_EnableIT_TXE

LL_I2S_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_I2S_DisableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Disable error IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_DisableIT_ERR

LL_I2S_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_I2S_DisableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Disable Rx buffer not empty IT.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_DisableIT_RXNE

LL_I2S_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_I2S_DisableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Disable Tx buffer empty IT.

Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_I2S_DisableIT_TXE

LL_I2S_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_ERR (SPI_TypeDef * SPIx)**
Function description	Check if ERR IT is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_IsEnabledIT_ERR

LL_I2S_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if RXNE IT is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_IsEnabledIT_RXNE

LL_I2S_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXE (SPI_TypeDef * SPIx)**
Function description	Check if TXE IT is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_I2S_IsEnabledIT_TXE

LL_I2S_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Rx.
Parameters	• **SPIx:** SPI Instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Rx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Tx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name	**__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)**
Function description	Read 16-Bits in data register.
Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• DR DR LL_I2S_ReceiveData16

LL_I2S_TransmitData16

Function name	**__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)**
Function description	Write 16-Bits in data register.
Parameters	• **SPIx:** SPI Instance • **TxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_I2S_TransmitData16

LL_I2S_DeInit

Function name	**ErrorStatus LL_I2S_DeInit (SPI_TypeDef * SPIx)**
Function description	De-initialize the SPI/I2S registers to their default reset values.
Parameters	• **SPIx:** SPI Instance
Return values	• **An:** ErrorStatus enumeration value:

- SUCCESS: SPI registers are de-initialized
- ERROR: SPI registers are not de-initialized

LL_I2S_Init

Function name **ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)**

Function description Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters
- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes
- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_I2S_StructInit

Function name **void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)**

Function description Set each LL_I2S_InitTypeDef field to default value.

Parameters
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values
- **None:**

LL_I2S_ConfigPrescaler

Function name **void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)**

Function description Set linear and parity prescaler.

Parameters
- **SPIx:** SPI Instance
- **PrescalerLinear:** value: Min_Data=0x02 and Max_Data=0xFF.
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values
- **None:**

Notes
- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).

55.3 I2S Firmware driver defines

55.3.1 I2S

Audio Frequency

LL_I2S_AUDIOFREQ_192K	Audio Frequency configuration 192000 Hz
LL_I2S_AUDIOFREQ_96K	Audio Frequency configuration 96000 Hz
LL_I2S_AUDIOFREQ_48K	Audio Frequency configuration 48000 Hz
LL_I2S_AUDIOFREQ_44K	Audio Frequency configuration 44100 Hz
LL_I2S_AUDIOFREQ_32K	Audio Frequency configuration 32000 Hz
LL_I2S_AUDIOFREQ_22K	Audio Frequency configuration 22050 Hz
LL_I2S_AUDIOFREQ_16K	Audio Frequency configuration 16000 Hz
LL_I2S_AUDIOFREQ_11K	Audio Frequency configuration 11025 Hz
LL_I2S_AUDIOFREQ_8K	Audio Frequency configuration 8000 Hz
LL_I2S_AUDIOFREQ_DEFAULT	Audio Freq not specified. Register I2SDIV = 2

Data format

LL_I2S_DATAFORMAT_16B	Data length 16 bits, Channel lenght 16bit
LL_I2S_DATAFORMAT_16B_EXTENDED	Data length 16 bits, Channel lenght 32bit
LL_I2S_DATAFORMAT_24B	Data length 24 bits, Channel lenght 32bit
LL_I2S_DATAFORMAT_32B	Data length 16 bits, Channel lenght 32bit

Get Flags Defines

LL_I2S_SR_RXNE	Rx buffer not empty flag
LL_I2S_SR_TXE	Tx buffer empty flag
LL_I2S_SR_BSY	Busy flag
LL_I2S_SR_UDR	Underrun flag
LL_I2S_SR_OVR	Overrun flag
LL_I2S_SR_FRE	TI mode frame format error flag

MCLK Output

LL_I2S_MCLK_OUTPUT_DISABLE	Master clock output is disabled
LL_I2S_MCLK_OUTPUT_ENABLE	Master clock output is enabled

Operation Mode

LL_I2S_MODE_SLAVE_TX	Slave Tx configuration
LL_I2S_MODE_SLAVE_RX	Slave Rx configuration
LL_I2S_MODE_MASTER_TX	Master Tx configuration
LL_I2S_MODE_MASTER_RX	Master Rx configuration

Clock Polarity

LL_I2S_POLARITY_LOW	Clock steady state is low level
LL_I2S_POLARITY_HIGH	Clock steady state is high level

Prescaler Factor

LL_I2S_PRESCALER_PARITY_EVEN	Odd factor: Real divider value is = I2SDIV * 2
LL_I2S_PRESCALER_PARITY_ODD	Odd factor: Real divider value is = (I2SDIV * 2)+1

I2s Standard

LL_I2S_STANDARD_PHILIPS	I2S standard philips
LL_I2S_STANDARD_MSB	MSB justified standard (left justified)
LL_I2S_STANDARD_LSB	LSB justified standard (right justified)
LL_I2S_STANDARD_PCM_SHORT	PCM standard, short frame synchronization
LL_I2S_STANDARD_PCM_LONG	PCM standard, long frame synchronization

Common Write and read registers Macros

LL_I2S_WriteReg

Description:

- Write a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2S_ReadReg

Description:

- Read a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be read

Return value:

- Register: value

56 LL IWDG Generic Driver

56.1 IWDG Firmware driver API description

56.1.1 Detailed description of functions

LL_IWDG_Enable

Function name	**__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)**
Function description	Start the Independent Watchdog.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Notes	• Except if the hardware watchdog option is selected
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name	**__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)**
Function description	Reloads IWDG counter with value defined in the reload register.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name	**__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)**
Function description	Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name	**__STATIC_INLINE void LL_IWDG_DisableWriteAccess**

(IWDG_TypeDef * IWDGx)

Function description: Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters:
- **IWDGx:** IWDG Instance

Return values:
- **None:**

Reference Manual to LL API cross reference:
- KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name: **__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)**

Function description: Select the prescaler of the IWDG.

Parameters:
- **IWDGx:** IWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256

Return values:
- **None:**

Reference Manual to LL API cross reference:
- PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name: **__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (IWDG_TypeDef * IWDGx)**

Function description: Get the selected prescaler of the IWDG.

Parameters:
- **IWDGx:** IWDG Instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256

Reference Manual to LL API cross reference:
- PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name	**__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)**
Function description	Specify the IWDG down-counter reload value.
Parameters	• **IWDGx:** IWDG Instance • **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name	**__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)**
Function description	Get the specified IWDG down-counter reload value.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **Value:** between Min_Data=0 and Max_Data=0x0FFF
Reference Manual to LL API cross reference:	• RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_IsActiveFlag_PVU

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (IWDG_TypeDef * IWDGx)**
Function description	Check if flag Prescaler Value Update is set or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (IWDG_TypeDef * IWDGx)**
Function description	Check if flag Reload Value Update is set or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsReady

Function name	**__STATIC_INLINE uint32_t LL_IWDG_IsReady (IWDG_TypeDef * IWDGx)**
Function description	Check if all flags Prescaler, Reload & Window Value Update are reset or not.
Parameters	• **IWDGx:** IWDG Instance
Return values	• **State:** of bits (1 or 0).
Reference Manual to LL API cross reference:	• SR PVU LL_IWDG_IsReady • SR RVU LL_IWDG_IsReady

56.2 IWDG Firmware driver defines

56.2.1 IWDG

Get Flags Defines

LL_IWDG_SR_PVU	Watchdog prescaler value update
LL_IWDG_SR_RVU	Watchdog counter reload value update

Prescaler Divider

LL_IWDG_PRESCALER_4	Divider by 4
LL_IWDG_PRESCALER_8	Divider by 8
LL_IWDG_PRESCALER_16	Divider by 16
LL_IWDG_PRESCALER_32	Divider by 32
LL_IWDG_PRESCALER_64	Divider by 64
LL_IWDG_PRESCALER_128	Divider by 128
LL_IWDG_PRESCALER_256	Divider by 256

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

57 LL PWR Generic Driver

57.1 PWR Firmware driver API description

57.1.1 Detailed description of functions

LL_PWR_EnableBkUpAccess

Function name	**__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)**
Function description	Enable access to the backup domain.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name	**__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)**
Function description	Disable access to the backup domain.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR DBP LL_PWR_DisableBkUpAccess

LL_PWR_IsEnabledBkUpAccess

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)**
Function description	Check if the backup domain is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR DBP LL_PWR_IsEnabledBkUpAccess

LL_PWR_SetRegulModeDS

Function name	**__STATIC_INLINE void LL_PWR_SetRegulModeDS (uint32_t RegulMode)**
Function description	Set voltage Regulator mode during deep sleep mode.
Parameters	• **RegulMode:** This parameter can be one of the following values: – LL_PWR_REGU_DSMODE_MAIN – LL_PWR_REGU_DSMODE_LOW_POWER
Return values	• **None:**

Reference Manual to LL API cross reference:	• CR LPDS LL_PWR_SetRegulModeDS

LL_PWR_GetRegulModeDS

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetRegulModeDS (void)**
Function description	Get voltage Regulator mode during deep sleep mode.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_REGU_DSMODE_MAIN – LL_PWR_REGU_DSMODE_LOW_POWER
Reference Manual to LL API cross reference:	• CR LPDS LL_PWR_GetRegulModeDS

LL_PWR_SetPowerMode

Function name	**__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t PDMode)**
Function description	Set Power Down mode when CPU enters deepsleep.
Parameters	• **PDMode:** This parameter can be one of the following values: – LL_PWR_MODE_STOP_MAINREGU – LL_PWR_MODE_STOP_LPREGU – LL_PWR_MODE_STANDBY
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR PDDS LL_PWR_SetPowerMode • • CR LPDS LL_PWR_SetPowerMode

LL_PWR_GetPowerMode

Function name	**__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)**
Function description	Get Power Down mode when CPU enters deepsleep.
Return values	• **Returned:** value can be one of the following values: – LL_PWR_MODE_STOP_MAINREGU – LL_PWR_MODE_STOP_LPREGU – LL_PWR_MODE_STANDBY
Reference Manual to LL API cross reference:	• CR PDDS LL_PWR_GetPowerMode • • CR LPDS LL_PWR_GetPowerMode

LL_PWR_SetPVDLevel

Function name	**__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)**
Function description	Configure the voltage threshold detected by the Power Voltage Detector.

Parameters
- **PVDLevel:** This parameter can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name **__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)**

Function description Get the voltage threshold detection.

Return values
- **Returned:** value can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:
- CR PLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name **__STATIC_INLINE void LL_PWR_EnablePVD (void)**

Function description Enable Power Voltage Detector.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR PVDE LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name **__STATIC_INLINE void LL_PWR_DisablePVD (void)**

Function description Disable Power Voltage Detector.

Return values
- **None:**

Reference Manual to LL API cross
- CR PVDE LL_PWR_DisablePVD

reference:

LL_PWR_IsEnabledPVD

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)**
Function description	Check if Power Voltage Detector is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR PVDE LL_PWR_IsEnabledPVD

LL_PWR_EnableWakeUpPin

Function name	**__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)**
Function description	Enable the WakeUp PINx functionality.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR EWUP LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name	**__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)**
Function description	Disable the WakeUp PINx functionality.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR EWUP LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)**
Function description	Check if the WakeUp PINx functionality is enabled.
Parameters	• **WakeUpPin:** This parameter can be one of the following values: – LL_PWR_WAKEUP_PIN1
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CSR EWUP LL_PWR_IsEnabledWakeUpPin

LL_PWR_IsActiveFlag_WU

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU (void)**
Function description	Get Wake-up Flag.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR WUF LL_PWR_IsActiveFlag_WU

LL_PWR_IsActiveFlag_SB

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)**
Function description	Get Standby Flag.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_PVDO

Function name	**__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)**
Function description	Indicate whether VDD voltage is below the selected PVD threshold.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_ClearFlag_SB

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)**
Function description	Clear Standby Flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CSBF LL_PWR_ClearFlag_SB

LL_PWR_ClearFlag_WU

Function name	**__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)**
Function description	Clear Wake-up Flags.

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CWUF LL_PWR_ClearFlag_WU

LL_PWR_DeInit

Function name	**ErrorStatus LL_PWR_DeInit (void)**
Function description	De-initialize the PWR registers to their default reset values.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: PWR registers are de-initialized – ERROR: not applicable

57.2 PWR Firmware driver defines

57.2.1 PWR

Clear Flags Defines

LL_PWR_CR_CSBF	Clear standby flag
LL_PWR_CR_CWUF	Clear wakeup flag

Get Flags Defines

LL_PWR_CSR_WUF	Wakeup flag
LL_PWR_CSR_SBF	Standby flag
LL_PWR_CSR_PVDO	Power voltage detector output flag
LL_PWR_CSR_EWUP1	Enable WKUP pin 1

Mode Power

LL_PWR_MODE_STOP_MAINREGU	Enter Stop mode when the CPU enters deepsleep
LL_PWR_MODE_STOP_LPREGU	Enter Stop mode (with low power Regulator ON) when the CPU enters deepsleep
LL_PWR_MODE_STANDBY	Enter Standby mode when the CPU enters deepsleep

Power Voltage Detector Level

LL_PWR_PVDLEVEL_0	Voltage threshold detected by PVD 2.2 V
LL_PWR_PVDLEVEL_1	Voltage threshold detected by PVD 2.3 V
LL_PWR_PVDLEVEL_2	Voltage threshold detected by PVD 2.4 V
LL_PWR_PVDLEVEL_3	Voltage threshold detected by PVD 2.5 V
LL_PWR_PVDLEVEL_4	Voltage threshold detected by PVD 2.6 V
LL_PWR_PVDLEVEL_5	Voltage threshold detected by PVD 2.7 V
LL_PWR_PVDLEVEL_6	Voltage threshold detected by PVD 2.8 V
LL_PWR_PVDLEVEL_7	Voltage threshold detected by PVD 2.9 V

Regulator Mode In Deep Sleep Mode

LL_PWR_REGU_DSMODE_MAIN	Voltage Regulator in main mode during deepsleep mode
LL_PWR_REGU_DSMODE_LOW_POWER	Voltage Regulator in low-power mode during deepsleep mode

Wakeup Pins

LL_PWR_WAKEUP_PIN1 WKUP pin 1 : PA0

Common write and read registers Macros

LL_PWR_WriteReg

Description:

- Write a value in PWR register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PWR_ReadReg

Description:

- Read a value in PWR register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

58 LL RCC Generic Driver

58.1 RCC Firmware driver registers structures

58.1.1 LL_RCC_ClocksTypeDef

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
  HCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
  PCLK1 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
  PCLK2 clock frequency

58.2 RCC Firmware driver API description

58.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name	**__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)**
Function description	Enable the Clock Security System.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR CSSON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name	**__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)**
Function description	Enable HSE external oscillator (HSE Bypass)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name	**__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)**

Function description	Disable HSE external oscillator (HSE Bypass)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSE_Enable (void)**
Function description	Enable HSE crystal oscillator (HSE ON)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSE_Disable (void)**
Function description	Disable HSE crystal oscillator (HSE ON)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)**
Function description	Check if HSE oscillator Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_HSI_Enable (void)**
Function description	Enable HSI oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name	**__STATIC_INLINE void LL_RCC_HSI_Disable (void)**
Function description	Disable HSI oscillator.

Return values	• **None:**
Reference Manual to LL API cross reference:	• CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)**
Function description	Check if HSI clock is ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_GetCalibration

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)**
Function description	Get HSI Calibration value.
Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0xFF
Notes	• When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value
Reference Manual to LL API cross reference:	• CR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name	**__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)**
Function description	Set HSI Calibration trimming.
Parameters	• **Value:** between Min_Data = 0x00 and Max_Data = 0x1F
Return values	• **None:**
Notes	• user-programmable trimming value that is added to the HSICAL • Default value is 16, which, when added to the HSICAL value, should trim the HSI to 16 MHz +/- 1 %
Reference Manual to LL API cross reference:	• CR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name	**__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)**
Function description	Get HSI Calibration trimming.

Return values	• **Between:** Min_Data = 0x00 and Max_Data = 0x1F
Reference Manual to LL API cross reference:	• CR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_LSE_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSE_Enable (void)**
Function description	Enable Low Speed External (LSE) crystal.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name	**__STATIC_INLINE void LL_RCC_LSE_Disable (void)**
Function description	Disable Low Speed External (LSE) crystal.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name	**__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)**
Function description	Enable external clock source (LSE bypass).
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name	**__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)**
Function description	Disable external clock source (LSE bypass).
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)**
Function description	Check if LSE oscillator Ready.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSI_Enable

Function name	**__STATIC_INLINE void LL_RCC_LSI_Enable (void)**
Function description	Enable LSI Oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR LSION LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name	**__STATIC_INLINE void LL_RCC_LSI_Disable (void)**
Function description	Disable LSI Oscillator.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name	**__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)**
Function description	Check if LSI is Ready.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_SetSysClkSource

Function name	**__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)**
Function description	Configure the system clock source.
Parameters	• **Source:** This parameter can be one of the following values: – LL_RCC_SYS_CLKSOURCE_HSI – LL_RCC_SYS_CLKSOURCE_HSE – LL_RCC_SYS_CLKSOURCE_PLL
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name **__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)**

Function description Get the system clock source.

Return values
- **Returned:** value can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_PLL

Reference Manual to LL API cross reference:
- CFGR SWS LL_RCC_GetSysClkSource

LL_RCC_SetAHBPrescaler

Function name **__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)**

Function description Set AHB prescaler.

Parameters
- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return values
- **None:**

Reference Manual to LL API cross reference:
- CFGR HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetAPB1Prescaler

Function name **__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)**

Function description Set APB1 prescaler.

Parameters
- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return values
- **None:**

Reference Manual to LL API cross
- CFGR PPRE1 LL_RCC_SetAPB1Prescaler

reference:

LL_RCC_SetAPB2Prescaler

Function name	**__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)**
Function description	Set APB2 prescaler.
Parameters	• **Prescaler:** This parameter can be one of the following values: – LL_RCC_APB2_DIV_1 – LL_RCC_APB2_DIV_2 – LL_RCC_APB2_DIV_4 – LL_RCC_APB2_DIV_8 – LL_RCC_APB2_DIV_16
Return values	• **None:**
Reference Manual to LL API cross reference:	• CFGR PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_GetAHBPrescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)**
Function description	Get AHB prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_SYSCLK_DIV_1 – LL_RCC_SYSCLK_DIV_2 – LL_RCC_SYSCLK_DIV_4 – LL_RCC_SYSCLK_DIV_8 – LL_RCC_SYSCLK_DIV_16 – LL_RCC_SYSCLK_DIV_64 – LL_RCC_SYSCLK_DIV_128 – LL_RCC_SYSCLK_DIV_256 – LL_RCC_SYSCLK_DIV_512
Reference Manual to LL API cross reference:	• CFGR HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetAPB1Prescaler

Function name	**__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)**
Function description	Get APB1 prescaler.
Return values	• **Returned:** value can be one of the following values: – LL_RCC_APB1_DIV_1 – LL_RCC_APB1_DIV_2 – LL_RCC_APB1_DIV_4 – LL_RCC_APB1_DIV_8 – LL_RCC_APB1_DIV_16
Reference Manual to	• CFGR PPRE1 LL_RCC_GetAPB1Prescaler

LL API cross reference:

LL_RCC_GetAPB2Prescaler

Function name **__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)**

Function description Get APB2 prescaler.

Return values
- **Returned:** value can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Reference Manual to LL API cross reference:
- CFGR PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_ConfigMCO

Function name **__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource)**

Function description Configure MCOx.

Parameters
- **MCOxSource:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_RCC_MCO1SOURCE_NOCLOCK
  - LL_RCC_MCO1SOURCE_SYSCLK
  - LL_RCC_MCO1SOURCE_HSI
  - LL_RCC_MCO1SOURCE_HSE
  - LL_RCC_MCO1SOURCE_PLLCLK_DIV_2
  - LL_RCC_MCO1SOURCE_PLL2CLK (*)
  - LL_RCC_MCO1SOURCE_PLLI2SCLK_DIV2 (*)
  - LL_RCC_MCO1SOURCE_EXT_HSE (*)
  - LL_RCC_MCO1SOURCE_PLLI2SCLK (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- CFGR MCO LL_RCC_ConfigMCO

LL_RCC_SetUSBClockSource

Function name **__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)**

Function description Configure USB clock source.

Parameters
- **USBxSource:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_RCC_USB_CLKSOURCE_PLL (*)
  - LL_RCC_USB_CLKSOURCE_PLL_DIV_1_5 (*)
  - LL_RCC_USB_CLKSOURCE_PLL_DIV_2 (*)

– LL_RCC_USB_CLKSOURCE_PLL_DIV_3 (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- CFGR OTGFSPRE LL_RCC_SetUSBClockSource
- CFGR USBPRE LL_RCC_SetUSBClockSource

LL_RCC_SetADCClockSource

Function name
__STATIC_INLINE void LL_RCC_SetADCClockSource (uint32_t ADCxSource)

Function description
Configure ADC clock source.

Parameters
- **ADCxSource:** This parameter can be one of the following values:
  – LL_RCC_ADC_CLKSRC_PCLK2_DIV_2
  – LL_RCC_ADC_CLKSRC_PCLK2_DIV_4
  – LL_RCC_ADC_CLKSRC_PCLK2_DIV_6
  – LL_RCC_ADC_CLKSRC_PCLK2_DIV_8

Return values
- **None:**

Reference Manual to LL API cross reference:
- CFGR ADCPRE LL_RCC_SetADCClockSource

LL_RCC_GetUSBClockSource

Function name
__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t USBx)

Function description
Get USBx clock source.

Parameters
- **USBx:** This parameter can be one of the following values:
  – LL_RCC_USB_CLKSOURCE

Return values
- **Returned:** value can be one of the following values: (*) value not defined in all devices
  – LL_RCC_USB_CLKSOURCE_PLL (*)
  – LL_RCC_USB_CLKSOURCE_PLL_DIV_1_5 (*)
  – LL_RCC_USB_CLKSOURCE_PLL_DIV_2 (*)
  – LL_RCC_USB_CLKSOURCE_PLL_DIV_3 (*)

Reference Manual to LL API cross reference:
- CFGR OTGFSPRE LL_RCC_GetUSBClockSource
- CFGR USBPRE LL_RCC_GetUSBClockSource

LL_RCC_GetADCClockSource

Function name
__STATIC_INLINE uint32_t LL_RCC_GetADCClockSource (uint32_t ADCx)

Function description
Get ADCx clock source.

Parameters
- **ADCx:** This parameter can be one of the following values:
  – LL_RCC_ADC_CLKSOURCE

Return values
- **Returned:** value can be one of the following values:

- LL_RCC_ADC_CLKSRC_PCLK2_DIV_2
- LL_RCC_ADC_CLKSRC_PCLK2_DIV_4
- LL_RCC_ADC_CLKSRC_PCLK2_DIV_6
- LL_RCC_ADC_CLKSRC_PCLK2_DIV_8

Reference Manual to LL API cross reference:
- CFGR ADCPRE LL_RCC_GetADCClockSource

LL_RCC_SetRTCClockSource

Function name: **__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)**

Function description: Set RTC Clock Source.

Parameters:
- **Source:** This parameter can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV128

Return values:
- **None:**

Notes:
- Once the RTC clock source has been selected, it cannot be changed any more unless the Backup domain is reset. The BDRST bit can be used to reset them.

Reference Manual to LL API cross reference:
- BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name: **__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)**

Function description: Get RTC Clock Source.

Return values:
- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV128

Reference Manual to LL API cross reference:
- BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name: **__STATIC_INLINE void LL_RCC_EnableRTC (void)**

Function description: Enable RTC.

Return values:
- **None:**

Reference Manual to LL API cross
- BDCR RTCEN LL_RCC_EnableRTC

reference:

LL_RCC_DisableRTC

Function name	**__STATIC_INLINE void LL_RCC_DisableRTC (void)**
Function description	Disable RTC.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)**
Function description	Check if RTC has been enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• BDCR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name	**__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)**
Function description	Force the Backup domain reset.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR BDRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name	**__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)**
Function description	Release the Backup domain reset.
Return values	• **None:**
Reference Manual to LL API cross reference:	• BDCR BDRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_PLL_Enable

Function name	**__STATIC_INLINE void LL_RCC_PLL_Enable (void)**
Function description	Enable PLL.
Return values	• **None:**
Reference Manual to	• CR PLLON LL_RCC_PLL_Enable

LL API cross reference:

LL_RCC_PLL_Disable

Function name **__STATIC_INLINE void LL_RCC_PLL_Disable (void)**

Function description Disable PLL.

Return values
- **None:**

Notes
- Cannot be disabled if the PLL clock is used as the system clock

Reference Manual to LL API cross reference:
- CR PLLON LL_RCC_PLL_Disable

LL_RCC_PLL_IsReady

Function name **__STATIC_INLINE uint32_t LL_RCC_PLL_IsReady (void)**

Function description Check if PLL Ready.

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CR PLLRDY LL_RCC_PLL_IsReady

LL_RCC_PLL_ConfigDomain_SYS

Function name **__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SYS (uint32_t Source, uint32_t PLLMul)**

Function description Configure PLL used for SYSCLK Domain.

Parameters
- **Source:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_RCC_PLLSOURCE_HSI_DIV_2
  - LL_RCC_PLLSOURCE_HSE_DIV_1
  - LL_RCC_PLLSOURCE_HSE_DIV_2 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_3 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_4 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_5 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_6 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_7 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_8 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_9 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_10 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_11 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_12 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_13 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_14 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_15 (*)
  - LL_RCC_PLLSOURCE_HSE_DIV_16 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_2 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_3 (*)

  - LL_RCC_PLLSOURCE_PLL2_DIV_4 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_5 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_6 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_7 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_8 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_9 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_10 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_11 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_12 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_13 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_14 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_15 (*)
  - LL_RCC_PLLSOURCE_PLL2_DIV_16 (*)
- **PLLMul:** This parameter can be one of the following values: (*) value not defined in all devices
  - LL_RCC_PLL_MUL_2 (*)
  - LL_RCC_PLL_MUL_3 (*)
  - LL_RCC_PLL_MUL_4
  - LL_RCC_PLL_MUL_5
  - LL_RCC_PLL_MUL_6
  - LL_RCC_PLL_MUL_7
  - LL_RCC_PLL_MUL_8
  - LL_RCC_PLL_MUL_9
  - LL_RCC_PLL_MUL_6_5 (*)
  - LL_RCC_PLL_MUL_10 (*)
  - LL_RCC_PLL_MUL_11 (*)
  - LL_RCC_PLL_MUL_12 (*)
  - LL_RCC_PLL_MUL_13 (*)
  - LL_RCC_PLL_MUL_14 (*)
  - LL_RCC_PLL_MUL_15 (*)
  - LL_RCC_PLL_MUL_16 (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- CFGR PLLSRC LL_RCC_PLL_ConfigDomain_SYS
- CFGR PLLXTPRE LL_RCC_PLL_ConfigDomain_SYS
- CFGR PLLMULL LL_RCC_PLL_ConfigDomain_SYS
- CFGR2 PREDIV1 LL_RCC_PLL_ConfigDomain_SYS
- CFGR2 PREDIV1SRC LL_RCC_PLL_ConfigDomain_SYS

LL_RCC_PLL_GetMainSource

Function name
__STATIC_INLINE uint32_t LL_RCC_PLL_GetMainSource (void)

Function description
Get the oscillator used as PLL clock source.

Return values
- **Returned:** value can be one of the following values: (*) value not defined in all devices
  - LL_RCC_PLLSOURCE_HSI_DIV_2
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_PLL2 (*)

Reference Manual to LL API cross
- CFGR PLLSRC LL_RCC_PLL_GetMainSource
- CFGR2 PREDIV1SRC LL_RCC_PLL_GetMainSource

reference:

LL_RCC_PLL_GetMultiplicator

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_GetMultiplicator (void)**
Function description	Get PLL multiplication Factor.
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices – LL_RCC_PLL_MUL_2 (*) – LL_RCC_PLL_MUL_3 (*) – LL_RCC_PLL_MUL_4 – LL_RCC_PLL_MUL_5 – LL_RCC_PLL_MUL_6 – LL_RCC_PLL_MUL_7 – LL_RCC_PLL_MUL_8 – LL_RCC_PLL_MUL_9 – LL_RCC_PLL_MUL_6_5 (*) – LL_RCC_PLL_MUL_10 (*) – LL_RCC_PLL_MUL_11 (*) – LL_RCC_PLL_MUL_12 (*) – LL_RCC_PLL_MUL_13 (*) – LL_RCC_PLL_MUL_14 (*) – LL_RCC_PLL_MUL_15 (*) – LL_RCC_PLL_MUL_16 (*)
Reference Manual to LL API cross reference:	• CFGR PLLMULL LL_RCC_PLL_GetMultiplicator

LL_RCC_PLL_GetPrediv

Function name	**__STATIC_INLINE uint32_t LL_RCC_PLL_GetPrediv (void)**
Function description	Get PREDIV1 division factor for the main PLL.
Return values	• **Returned:** value can be one of the following values: (*) value not defined in all devices – LL_RCC_PREDIV_DIV_1 – LL_RCC_PREDIV_DIV_2 – LL_RCC_PREDIV_DIV_3 (*) – LL_RCC_PREDIV_DIV_4 (*) – LL_RCC_PREDIV_DIV_5 (*) – LL_RCC_PREDIV_DIV_6 (*) – LL_RCC_PREDIV_DIV_7 (*) – LL_RCC_PREDIV_DIV_8 (*) – LL_RCC_PREDIV_DIV_9 (*) – LL_RCC_PREDIV_DIV_10 (*) – LL_RCC_PREDIV_DIV_11 (*) – LL_RCC_PREDIV_DIV_12 (*) – LL_RCC_PREDIV_DIV_13 (*) – LL_RCC_PREDIV_DIV_14 (*) – LL_RCC_PREDIV_DIV_15 (*)

– LL_RCC_PREDIV_DIV_16 (*)

Notes
- They can be written only when the PLL is disabled

Reference Manual to LL API cross reference:
- CFGR2 PREDIV1 LL_RCC_PLL_GetPrediv
- CFGR2 PLLXTPRE LL_RCC_PLL_GetPrediv

LL_RCC_ClearFlag_LSIRDY

Function name **__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)**

Function description Clear LSI ready interrupt flag.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CIR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name **__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)**

Function description Clear LSE ready interrupt flag.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CIR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_HSIRDY

Function name **__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)**

Function description Clear HSI ready interrupt flag.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CIR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name **__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)**

Function description Clear HSE ready interrupt flag.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CIR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_PLLRDY

Function name **__STATIC_INLINE void LL_RCC_ClearFlag_PLLRDY (void)**

Function description Clear PLL ready interrupt flag.

Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR PLLRDYC LL_RCC_ClearFlag_PLLRDY

LL_RCC_ClearFlag_HSECSS

Function name	**__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)**
Function description	Clear Clock security system interrupt flag.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR CSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_IsActiveFlag_LSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)**
Function description	Check if LSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)**
Function description	Check if LSE ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_HSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)**
Function description	Check if HSI ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY**

(void)

Function description	Check if HSE ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_PLLRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLRDY (void)**
Function description	Check if PLL ready interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR PLLRDYF LL_RCC_IsActiveFlag_PLLRDY

LL_RCC_IsActiveFlag_HSECSS

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)**
Function description	Check if Clock security system interrupt occurred or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR CSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_IWDGRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)**
Function description	Check if RCC flag Independent Watchdog reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)**
Function description	Check if RCC flag Low Power reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_PINRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)**
Function description	Check if RCC flag Pin reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_PORRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST (void)**
Function description	Check if RCC flag POR/PDR reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR PORRSTF LL_RCC_IsActiveFlag_PORRST

LL_RCC_IsActiveFlag_SFTRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)**
Function description	Check if RCC flag Software reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_WWDGRST

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)**
Function description	Check if RCC flag Window Watchdog reset is set or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_ClearResetFlags

Function name	**__STATIC_INLINE void LL_RCC_ClearResetFlags (void)**
Function description	Set RMVF bit to clear the reset flags.
Return values	• **None:**
Reference Manual to LL API cross	• CSR RMVF LL_RCC_ClearResetFlags

reference:

LL_RCC_EnableIT_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)**
Function description	Enable LSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)**
Function description	Enable LSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)**
Function description	Enable HSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)**
Function description	Enable HSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_PLLRDY

Function name	**__STATIC_INLINE void LL_RCC_EnableIT_PLLRDY (void)**
Function description	Enable PLL ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR PLLRDYIE LL_RCC_EnableIT_PLLRDY

LL_RCC_DisableIT_LSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)**
Function description	Disable LSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)**
Function description	Disable LSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_HSIRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)**
Function description	Disable HSI ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)**
Function description	Disable HSE ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_PLLRDY

Function name	**__STATIC_INLINE void LL_RCC_DisableIT_PLLRDY (void)**
Function description	Disable PLL ready interrupt.
Return values	• **None:**
Reference Manual to LL API cross reference:	• CIR PLLRDYIE LL_RCC_DisableIT_PLLRDY

LL_RCC_IsEnabledIT_LSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)**
Function description	Checks if LSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)**
Function description	Checks if LSE ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_HSIRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)**
Function description	Checks if HSI ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

LL_RCC_IsEnabledIT_HSERDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)**
Function description	Checks if HSE ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CIR HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_PLLRDY

Function name	**__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLRDY (void)**
Function description	Checks if PLL ready interrupt source is enabled or disabled.
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CIR PLLRDYIE LL_RCC_IsEnabledIT_PLLRDY

LL_RCC_DeInit

Function name **ErrorStatus LL_RCC_DeInit (void)**

Function description Reset the RCC clock configuration to the default reset state.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: RCC registers are de-initialized
  - ERROR: not applicable

Notes
- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE PLL, PLL2, PLL3 OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name **void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)**

Function description Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.

Parameters
- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values
- **None:**

Notes
- Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSBClockFreq

Function name **uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)**

Function description Return USBx clock frequency.

Parameters
- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values
- **USB:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI), HSE or PLL is not ready

LL_RCC_GetADCClockFreq

Function name **uint32_t LL_RCC_GetADCClockFreq (uint32_t ADCxSource)**

Function description Return ADCx clock frequency.

Parameters
- **ADCxSource:** This parameter can be one of the following values:
  – LL_RCC_ADC_CLKSOURCE

Return values
- **ADC:** clock frequency (in Hz)

58.3 RCC Firmware driver defines

58.3.1 RCC

Peripheral ADC get clock source

LL_RCC_ADC_CLKSOURCE ADC Clock source selection

Peripheral ADC clock source selection

LL_RCC_ADC_CLKSRC_PCLK2_DIV_2

LL_RCC_ADC_CLKSRC_PCLK2_DIV_4

LL_RCC_ADC_CLKSRC_PCLK2_DIV_6

LL_RCC_ADC_CLKSRC_PCLK2_DIV_8

APB low-speed prescaler (APB1)

LL_RCC_APB1_DIV_1 HCLK not divided

LL_RCC_APB1_DIV_2 HCLK divided by 2

LL_RCC_APB1_DIV_4 HCLK divided by 4

LL_RCC_APB1_DIV_8 HCLK divided by 8

LL_RCC_APB1_DIV_16 HCLK divided by 16

APB high-speed prescaler (APB2)

LL_RCC_APB2_DIV_1 HCLK not divided

LL_RCC_APB2_DIV_2 HCLK divided by 2

LL_RCC_APB2_DIV_4 HCLK divided by 4

LL_RCC_APB2_DIV_8 HCLK divided by 8

LL_RCC_APB2_DIV_16 HCLK divided by 16

Clear Flags Defines

LL_RCC_CIR_LSIRDYC LSI Ready Interrupt Clear

LL_RCC_CIR_LSERDYC LSE Ready Interrupt Clear

LL_RCC_CIR_HSIRDYC HSI Ready Interrupt Clear

LL_RCC_CIR_HSERDYC HSE Ready Interrupt Clear

LL_RCC_CIR_PLLRDYC PLL Ready Interrupt Clear

LL_RCC_CIR_PLL3RDYC PLL3(PLLI2S) Ready Interrupt Clear

LL_RCC_CIR_PLL2RDYC PLL2 Ready Interrupt Clear

LL_RCC_CIR_CSSC Clock Security System Interrupt Clear

Get Flags Defines

LL_RCC_CIR_LSIRDYF LSI Ready Interrupt flag

LL_RCC_CIR_LSERDYF	LSE Ready Interrupt flag
LL_RCC_CIR_HSIRDYF	HSI Ready Interrupt flag
LL_RCC_CIR_HSERDYF	HSE Ready Interrupt flag
LL_RCC_CIR_PLLRDYF	PLL Ready Interrupt flag
LL_RCC_CIR_PLL3RDYF	PLL3(PLLI2S) Ready Interrupt flag
LL_RCC_CIR_PLL2RDYF	PLL2 Ready Interrupt flag
LL_RCC_CIR_CSSF	Clock Security System Interrupt flag
LL_RCC_CSR_PINRSTF	PIN reset flag
LL_RCC_CSR_PORRSTF	POR/PDR reset flag
LL_RCC_CSR_SFTRSTF	Software Reset flag
LL_RCC_CSR_IWDGRSTF	Independent Watchdog reset flag
LL_RCC_CSR_WWDGRSTF	Window watchdog reset flag
LL_RCC_CSR_LPWRRSTF	Low-Power reset flag

IT Defines

LL_RCC_CIR_LSIRDYIE	LSI Ready Interrupt Enable
LL_RCC_CIR_LSERDYIE	LSE Ready Interrupt Enable
LL_RCC_CIR_HSIRDYIE	HSI Ready Interrupt Enable
LL_RCC_CIR_HSERDYIE	HSE Ready Interrupt Enable
LL_RCC_CIR_PLLRDYIE	PLL Ready Interrupt Enable
LL_RCC_CIR_PLL3RDYIE	PLL3(PLLI2S) Ready Interrupt Enable
LL_RCC_CIR_PLL2RDYIE	PLL2 Ready Interrupt Enable

MCO1 SOURCE selection

LL_RCC_MCO1SOURCE_NOCLOCK	MCO output disabled, no clock on MCO
LL_RCC_MCO1SOURCE_SYSCLK	SYSCLK selection as MCO source
LL_RCC_MCO1SOURCE_HSI	HSI selection as MCO source
LL_RCC_MCO1SOURCE_HSE	HSE selection as MCO source
LL_RCC_MCO1SOURCE_PLLCLK_DIV_2	PLL clock divided by 2

Oscillator Values adaptation

HSE_VALUE	Value of the HSE oscillator in Hz
HSI_VALUE	Value of the HSI oscillator in Hz
LSE_VALUE	Value of the LSE oscillator in Hz
LSI_VALUE	Value of the LSI oscillator in Hz

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO	No clock enabled for the peripheral
LL_RCC_PERIPH_FREQUENCY_NA	Frequency cannot be provided as external clock

PLL SOURCE

LL_RCC_PLLSOURCE_HSI_DIV_2	HSI clock divided by 2 selected as PLL entry clock source
LL_RCC_PLLSOURCE_HSE	HSE/PREDIV1 clock selected as PLL entry clock source
LL_RCC_PLLSOURCE_HSE_DIV_1	HSE clock selected as PLL entry clock source
LL_RCC_PLLSOURCE_HSE_DIV_2	HSE/2 clock selected as PLL entry clock source

PLL Multiplicator factor

LL_RCC_PLL_MUL_2	PLL input clock*2
LL_RCC_PLL_MUL_3	PLL input clock*3
LL_RCC_PLL_MUL_4	PLL input clock*4
LL_RCC_PLL_MUL_5	PLL input clock*5
LL_RCC_PLL_MUL_6	PLL input clock*6
LL_RCC_PLL_MUL_7	PLL input clock*7
LL_RCC_PLL_MUL_8	PLL input clock*8
LL_RCC_PLL_MUL_9	PLL input clock*9
LL_RCC_PLL_MUL_10	PLL input clock*10
LL_RCC_PLL_MUL_11	PLL input clock*11
LL_RCC_PLL_MUL_12	PLL input clock*12
LL_RCC_PLL_MUL_13	PLL input clock*13
LL_RCC_PLL_MUL_14	PLL input clock*14
LL_RCC_PLL_MUL_15	PLL input clock*15
LL_RCC_PLL_MUL_16	PLL input clock*16

PREDIV Division factor

LL_RCC_PREDIV_DIV_1	HSE divider clock clock not divided
LL_RCC_PREDIV_DIV_2	HSE divider clock divided by 2 for PLL entry

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE	No clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_LSE	LSE oscillator clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_LSI	LSI oscillator clock used as RTC clock
LL_RCC_RTC_CLKSOURCE_HSE_DIV128	HSE oscillator clock divided by 128 used as RTC clock

AHB prescaler

LL_RCC_SYSCLK_DIV_1	SYSCLK not divided
LL_RCC_SYSCLK_DIV_2	SYSCLK divided by 2
LL_RCC_SYSCLK_DIV_4	SYSCLK divided by 4
LL_RCC_SYSCLK_DIV_8	SYSCLK divided by 8
LL_RCC_SYSCLK_DIV_16	SYSCLK divided by 16

LL_RCC_SYSCLK_DIV_64	SYSCLK divided by 64
LL_RCC_SYSCLK_DIV_128	SYSCLK divided by 128
LL_RCC_SYSCLK_DIV_256	SYSCLK divided by 256
LL_RCC_SYSCLK_DIV_512	SYSCLK divided by 512

System clock switch

LL_RCC_SYS_CLKSOURCE_HSI	HSI selection as system clock
LL_RCC_SYS_CLKSOURCE_HSE	HSE selection as system clock
LL_RCC_SYS_CLKSOURCE_PLL	PLL selection as system clock

System clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_HSI	HSI used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_HSE	HSE used as system clock
LL_RCC_SYS_CLKSOURCE_STATUS_PLL	PLL used as system clock

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE	USB Clock source selection

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_PLL	PLL clock is not divided
LL_RCC_USB_CLKSOURCE_PLL_DIV_1_5	PLL clock is divided by 1.5

Calculate frequencies

__LL_RCC_CALC_PLLCLK_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE div Prediv1 or div 2 / HSI div 2)
- __PLLMUL__: This parameter can be one of the following values:
  - LL_RCC_PLL_MUL_2
  - LL_RCC_PLL_MUL_3
  - LL_RCC_PLL_MUL_4
  - LL_RCC_PLL_MUL_5
  - LL_RCC_PLL_MUL_6
  - LL_RCC_PLL_MUL_7
  - LL_RCC_PLL_MUL_8
  - LL_RCC_PLL_MUL_9
  - LL_RCC_PLL_MUL_10
  - LL_RCC_PLL_MUL_11
  - LL_RCC_PLL_MUL_12
  - LL_RCC_PLL_MUL_13
  - LL_RCC_PLL_MUL_14
  - LL_RCC_PLL_MUL_15
  - LL_RCC_PLL_MUL_16

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_FREQ (HSE_VALUE / (LL_RCC_PLL_GetPrediv () + 1), LL_RCC_PLL_GetMultiplicator ());

__LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency (based on HSE/HSI/PLLCLK)
- __AHBPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

Notes:

- : __AHBPRESCALER__ be retrieved by LL_RCC_GetAHBPrescaler ex: __LL_RCC_CALC_HCLK_FREQ(LL_RCC_GetAHBPrescaler())

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

Notes:

- : __APB1PRESCALER__ be retrieved by LL_RCC_GetAPB1Prescaler ex: __LL_RCC_CALC_PCLK1_FREQ(LL_RCC_GetAP

B1Prescaler())

__LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

Notes:

- : __APB2PRESCALER__ be retrieved by LL_RCC_GetAPB2Prescaler ex: __LL_RCC_CALC_PCLK2_FREQ(LL_RCC_GetAPB2Prescaler())

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

59 LL RTC Generic Driver

59.1 RTC Firmware driver registers structures

59.1.1 LL_RTC_InitTypeDef

Data Fields

- ***uint32_t AsynchPrescaler***
- ***uint32_t OutPutSource***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFFThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::OutPutSource***
  Specifies which signal will be routed to the RTC Tamper pin. This parameter can be a value of ***LL_RTC_Output_Source***This feature can be modified afterwards using unitary function **LL_RTC_SetOutputSource()**.

59.1.2 LL_RTC_TimeTypeDef

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint8_t LL_RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 23
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59

59.1.3 LL_RTC_AlarmTypeDef

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members.

59.2 RTC Firmware driver API description

59.2.1 Detailed description of functions

LL_RTC_SetAsynchPrescaler

Function name	**__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)**
Function description	Set Asynchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance • **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0xFFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• PRLH PRL LL_RTC_SetAsynchPrescaler • • PRLL PRL LL_RTC_SetAsynchPrescaler •

LL_RTC_GetDivider

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetDivider (RTC_TypeDef * RTCx)**
Function description	Get Asynchronous prescaler factor.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data = 0 and Max_Data = 0xFFFFF
Reference Manual to LL API cross reference:	• DIVH DIV LL_RTC_GetDivider • • DIVL DIV LL_RTC_GetDivider •

LL_RTC_SetOutputSource

Function name	**__STATIC_INLINE void LL_RTC_SetOutputSource (BKP_TypeDef * BKPx, uint32_t OutputSource)**
Function description	Set Output Source.
Parameters	• **BKPx:** BKP Instance • **OutputSource:** This parameter can be one of the following values: – LL_RTC_CALIB_OUTPUT_NONE – LL_RTC_CALIB_OUTPUT_RTCCLOCK – LL_RTC_CALIB_OUTPUT_ALARM – LL_RTC_CALIB_OUTPUT_SECOND
Return values	• **None:**
Reference Manual to LL API cross reference:	• RTCCR CCO LL_RTC_SetOutputSource • RTCCR ASOE LL_RTC_SetOutputSource • RTCCR ASOS LL_RTC_SetOutputSource

LL_RTC_GetOutPutSource

Function name	**__STATIC_INLINE uint32_t LL_RTC_GetOutPutSource (BKP_TypeDef * BKPx)**
Function description	Get Output Source.
Parameters	• **BKPx:** BKP Instance
Return values	• **Returned:** value can be one of the following values: – LL_RTC_CALIB_OUTPUT_NONE – LL_RTC_CALIB_OUTPUT_RTCCLOCK – LL_RTC_CALIB_OUTPUT_ALARM – LL_RTC_CALIB_OUTPUT_SECOND
Reference Manual to LL API cross reference:	• RTCCR CCO LL_RTC_GetOutPutSource • RTCCR ASOE LL_RTC_GetOutPutSource • RTCCR ASOS LL_RTC_GetOutPutSource

LL_RTC_EnableWriteProtection

Function name	**__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)**
Function description	Enable the write protection for RTC registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRL CNF LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name	**__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)**
Function description	Disable the write protection for RTC registers.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRL RTC_CRL_CNF LL_RTC_DisableWriteProtection

LL_RTC_TIME_Set

Function name	**__STATIC_INLINE void LL_RTC_TIME_Set (RTC_TypeDef * RTCx, uint32_t TimeCounter)**
Function description	Set time counter in BCD format.
Parameters	• **RTCx:** RTC Instance • **TimeCounter:** Value between Min_Data=0x00 and Max_Data=0xFFFFF
Return values	• **None:**

Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before. • It can be written in initialization mode only (LL_RTC_EnterInitMode function)
Reference Manual to LL API cross reference:	• CNTH CNT LL_RTC_TIME_Set • CNTL CNT LL_RTC_TIME_Set •

LL_RTC_TIME_Get

Function name	**__STATIC_INLINE uint32_t LL_RTC_TIME_Get (RTC_TypeDef * RTCx)**
Function description	Get time counter in BCD format.
Parameters	• **RTCx:** RTC Instance
Return values	• **Value:** between Min_Data = 0 and Max_Data = 0xFFFFF
Reference Manual to LL API cross reference:	• CNTH CNT LL_RTC_TIME_Get • CNTL CNT LL_RTC_TIME_Get •

LL_RTC_ALARM_Set

Function name	**__STATIC_INLINE void LL_RTC_ALARM_Set (RTC_TypeDef * RTCx, uint32_t AlarmCounter)**
Function description	Set Alarm Counter.
Parameters	• **RTCx:** RTC Instance • **AlarmCounter:** Value between Min_Data=0x00 and Max_Data=0xFFFFF
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• ALRH ALR LL_RTC_ALARM_Set • • ALRL ALR LL_RTC_ALARM_Set •

LL_RTC_ALARM_Get

Function name	**__STATIC_INLINE uint32_t LL_RTC_ALARM_Get (RTC_TypeDef * RTCx)**
Function description	Get Alarm Counter.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross	• ALRH ALR LL_RTC_ALARM_Get •

reference:
- ALRL ALR LL_RTC_ALARM_Get
-

LL_RTC_TAMPER_Enable

Function name **__STATIC_INLINE void LL_RTC_TAMPER_Enable (BKP_TypeDef * BKPx)**

Function description Enable RTC_TAMPx input detection.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR TPE LL_RTC_TAMPER_Enable
-

LL_RTC_TAMPER_Disable

Function name **__STATIC_INLINE void LL_RTC_TAMPER_Disable (BKP_TypeDef * BKPx)**

Function description Disable RTC_TAMPx Tamper.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR TPE LL_RTC_TAMPER_Disable
-

LL_RTC_TAMPER_SetActiveLevel

Function name **__STATIC_INLINE void LL_RTC_TAMPER_SetActiveLevel (BKP_TypeDef * BKPx, uint32_t Tamper)**

Function description Enable Active level for Tamper input.

Parameters
- **BKPx:** BKP Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ACTIVELEVEL_LOW
  - LL_RTC_TAMPER_ACTIVELEVEL_HIGH

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR TPAL LL_RTC_TAMPER_SetActiveLevel
-

LL_RTC_TAMPER_GetActiveLevel

Function name **__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetActiveLevel (BKP_TypeDef * BKPx)**

Function description Disable Active level for Tamper input.

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR TPAL LL_RTC_TAMPER_SetActiveLevel
-

LL_RTC_BKP_SetRegister

Function name **__STATIC_INLINE void LL_RTC_BKP_SetRegister (BKP_TypeDef * BKPx, uint32_t BackupRegister, uint32_t Data)**

Function description Writes a data in a specified RTC Backup data register.

Parameters

- **BKPx:** BKP Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11 (*)
  - LL_RTC_BKP_DR12 (*)
  - LL_RTC_BKP_DR13 (*)
  - LL_RTC_BKP_DR14 (*)
  - LL_RTC_BKP_DR15 (*)
  - LL_RTC_BKP_DR16 (*)
  - LL_RTC_BKP_DR17 (*)
  - LL_RTC_BKP_DR18 (*)
  - LL_RTC_BKP_DR19 (*)
  - LL_RTC_BKP_DR20 (*)
  - LL_RTC_BKP_DR21 (*)
  - LL_RTC_BKP_DR22 (*)
  - LL_RTC_BKP_DR23 (*)
  - LL_RTC_BKP_DR24 (*)
  - LL_RTC_BKP_DR25 (*)
  - LL_RTC_BKP_DR26 (*)
  - LL_RTC_BKP_DR27 (*)
  - LL_RTC_BKP_DR28 (*)
  - LL_RTC_BKP_DR29 (*)
  - LL_RTC_BKP_DR30 (*)
  - LL_RTC_BKP_DR31 (*)
  - LL_RTC_BKP_DR32 (*)
  - LL_RTC_BKP_DR33 (*)
  - LL_RTC_BKP_DR34 (*)
  - LL_RTC_BKP_DR35 (*)
  - LL_RTC_BKP_DR36 (*)
  - LL_RTC_BKP_DR37 (*)
  - LL_RTC_BKP_DR38 (*)
  - LL_RTC_BKP_DR39 (*)
  - LL_RTC_BKP_DR40 (*)
  - LL_RTC_BKP_DR41 (*)
  - LL_RTC_BKP_DR42 (*) (*) value not defined in all devices.

- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values
- **None:**

Reference Manual to LL API cross reference:
- BKPDR DR LL_RTC_BKP_SetRegister

LL_RTC_BKP_GetRegister

Function name
__STATIC_INLINE uint32_t LL_RTC_BKP_GetRegister (BKP_TypeDef * BKPx, uint32_t BackupRegister)

Function description
Reads data from the specified RTC Backup data Register.

Parameters
- **BKPx:** BKP Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11 (*)
  - LL_RTC_BKP_DR12 (*)
  - LL_RTC_BKP_DR13 (*)
  - LL_RTC_BKP_DR14 (*)
  - LL_RTC_BKP_DR15 (*)
  - LL_RTC_BKP_DR16 (*)
  - LL_RTC_BKP_DR17 (*)
  - LL_RTC_BKP_DR18 (*)
  - LL_RTC_BKP_DR19 (*)
  - LL_RTC_BKP_DR20 (*)
  - LL_RTC_BKP_DR21 (*)
  - LL_RTC_BKP_DR22 (*)
  - LL_RTC_BKP_DR23 (*)
  - LL_RTC_BKP_DR24 (*)
  - LL_RTC_BKP_DR25 (*)
  - LL_RTC_BKP_DR26 (*)
  - LL_RTC_BKP_DR27 (*)
  - LL_RTC_BKP_DR28 (*)
  - LL_RTC_BKP_DR29 (*)
  - LL_RTC_BKP_DR30 (*)
  - LL_RTC_BKP_DR31 (*)
  - LL_RTC_BKP_DR32 (*)
  - LL_RTC_BKP_DR33 (*)
  - LL_RTC_BKP_DR34 (*)
  - LL_RTC_BKP_DR35 (*)
  - LL_RTC_BKP_DR36 (*)

  - LL_RTC_BKP_DR37 (*)
  - LL_RTC_BKP_DR38 (*)
  - LL_RTC_BKP_DR39 (*)
  - LL_RTC_BKP_DR40 (*)
  - LL_RTC_BKP_DR41 (*)
  - LL_RTC_BKP_DR42 (*)

Return values
- **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:
- BKPDR DR LL_RTC_BKP_GetRegister

LL_RTC_CAL_SetCoarseDigital

Function name
__STATIC_INLINE void LL_RTC_CAL_SetCoarseDigital (BKP_TypeDef * BKPx, uint32_t Value)

Function description
Set the coarse digital calibration.

Parameters
- **BKPx:** RTC Instance
- **Value:** value of coarse calibration expressed in ppm (coded on 5 bits)

Return values
- **None:**

Notes
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnterInitMode function)
- This Calibration value should be between 0 and 121 when using positive sign with a 4-ppm step.

Reference Manual to LL API cross reference:
- RTCCR CAL LL_RTC_CAL_SetCoarseDigital
-

LL_RTC_CAL_GetCoarseDigital

Function name
__STATIC_INLINE uint32_t LL_RTC_CAL_GetCoarseDigital (BKP_TypeDef * BKPx)

Function description
Get the coarse digital calibration value.

Parameters
- **BKPx:** BKP Instance

Return values
- **value:** of coarse calibration expressed in ppm (coded on 5 bits)

Reference Manual to LL API cross reference:
- RTCCR CAL LL_RTC_CAL_SetCoarseDigital
-

LL_RTC_IsActiveFlag_TAMPI

Function name
__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMPI (BKP_TypeDef * BKPx)

Function description	Get RTC_TAMPI Interruption detection flag.
Parameters	• **BKPx:** BKP Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR TIF LL_RTC_IsActiveFlag_TAMPI

LL_RTC_ClearFlag_TAMPI

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMPI (BKP_TypeDef * BKPx)**
Function description	Clear RTC_TAMP Interruption detection flag.
Parameters	• **BKPx:** BKP Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR CTI LL_RTC_ClearFlag_TAMPI

LL_RTC_IsActiveFlag_TAMPE

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMPE (BKP_TypeDef * BKPx)**
Function description	Get RTC_TAMPE Event detection flag.
Parameters	• **BKPx:** BKP Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR TEF LL_RTC_IsActiveFlag_TAMPE

LL_RTC_ClearFlag_TAMPE

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_TAMPE (BKP_TypeDef * BKPx)**
Function description	Clear RTC_TAMPE Even detection flag.
Parameters	• **BKPx:** BKP Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR CTE LL_RTC_ClearFlag_TAMPE

LL_RTC_IsActiveFlag_ALR

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALR (RTC_TypeDef * RTCx)**
Function description	Get Alarm flag.

Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRL ALRF LL_RTC_IsActiveFlag_ALR

LL_RTC_ClearFlag_ALR

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_ALR (RTC_TypeDef * RTCx)**
Function description	Clear Alarm flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRL ALRF LL_RTC_ClearFlag_ALR

LL_RTC_IsActiveFlag_RS

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)**
Function description	Get Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRL RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)**
Function description	Clear Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRL RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_OW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_OW (RTC_TypeDef * RTCx)**
Function description	Get Registers OverFlow flag.
Parameters	• **RTCx:** RTC Instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRL OWF LL_RTC_IsActiveFlag_OW

LL_RTC_ClearFlag_OW

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_OW (RTC_TypeDef * RTCx)**
Function description	Clear Registers OverFlow flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRL OWF LL_RTC_ClearFlag_OW

LL_RTC_IsActiveFlag_SEC

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SEC (RTC_TypeDef * RTCx)**
Function description	Get Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRL SECF LL_RTC_IsActiveFlag_SEC

LL_RTC_ClearFlag_SEC

Function name	**__STATIC_INLINE void LL_RTC_ClearFlag_SEC (RTC_TypeDef * RTCx)**
Function description	Clear Registers synchronization flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRL SECF LL_RTC_ClearFlag_SEC

LL_RTC_IsActiveFlag_RTOF

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RTOF (RTC_TypeDef * RTCx)**
Function description	Get RTC Operation OFF status flag.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CRL RTOFF LL_RTC_IsActiveFlag_RTOF

LL_RTC_EnableIT_ALR

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_ALR (RTC_TypeDef * RTCx)**
Function description	Enable Alarm interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CRH ALRIE LL_RTC_EnableIT_ALR

LL_RTC_DisableIT_ALR

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_ALR (RTC_TypeDef * RTCx)**
Function description	Disable Alarm interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CRH ALRIE LL_RTC_DisableIT_ALR

LL_RTC_IsEnabledIT_ALR

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALR (RTC_TypeDef * RTCx)**
Function description	Check if Alarm interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRH ALRIE LL_RTC_IsEnabledIT_ALR

LL_RTC_EnableIT_SEC

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_SEC (RTC_TypeDef * RTCx)**
Function description	Enable Second Interrupt interrupt.

Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CRH SECIE LL_RTC_EnableIT_SEC

LL_RTC_DisableIT_SEC

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_SEC (RTC_TypeDef * RTCx)**
Function description	Disable Second interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CRH SECIE LL_RTC_DisableIT_SEC

LL_RTC_IsEnabledIT_SEC

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_SEC (RTC_TypeDef * RTCx)**
Function description	Check if Second interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRH SECIE LL_RTC_IsEnabledIT_SEC

LL_RTC_EnableIT_OW

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_OW (RTC_TypeDef * RTCx)**
Function description	Enable OverFlow interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CRH OWIE LL_RTC_EnableIT_OW

LL_RTC_DisableIT_OW

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_OW (RTC_TypeDef * RTCx)**
Function description	Disable OverFlow interrupt.
Parameters	• **RTCx:** RTC Instance
Return values	• **None:**
Notes	• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
Reference Manual to LL API cross reference:	• CRH OWIE LL_RTC_DisableIT_OW

LL_RTC_IsEnabledIT_OW

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_OW (RTC_TypeDef * RTCx)**
Function description	Check if OverFlow interrupt is enabled or not.
Parameters	• **RTCx:** RTC Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CRH OWIE LL_RTC_IsEnabledIT_OW

LL_RTC_EnableIT_TAMP

Function name	**__STATIC_INLINE void LL_RTC_EnableIT_TAMP (BKP_TypeDef * BKPx)**
Function description	Enable Tamper interrupt.
Parameters	• **BKPx:** BKP Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR TPIE LL_RTC_EnableIT_TAMP

LL_RTC_DisableIT_TAMP

Function name	**__STATIC_INLINE void LL_RTC_DisableIT_TAMP (BKP_TypeDef * BKPx)**
Function description	Disable Tamper interrupt.
Parameters	• **BKPx:** BKP Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CSR TPIE LL_RTC_EnableIT_TAMP

LL_RTC_IsEnabledIT_TAMP

Function name	**__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP (BKP_TypeDef * BKPx)**
Function description	Check if all the TAMPER interrupts are enabled or not.
Parameters	• **BKPx:** BKP Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR TPIE LL_RTC_IsEnabledIT_TAMP

LL_RTC_DeInit

Function name	**ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)**
Function description	De-Initializes the RTC registers to their default reset values.
Parameters	• **RTCx:** RTC Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are de-initialized – ERROR: RTC registers are not de-initialized
Notes	• This function doesn't reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name	**ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)**
Function description	Initializes the RTC registers according to the specified parameters in RTC_InitStruct.
Parameters	• **RTCx:** RTC Instance • **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC registers are initialized – ERROR: RTC registers are not initialized
Notes	• The RTC Prescaler register is write protected and can be written in initialization mode only. • the user should call LL_RTC_StructInit() or the structure of Prescaler need to be initialized before RTC init()

LL_RTC_StructInit

Function name	**void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)**
Function description	Set each LL_RTC_InitTypeDef field to default value.
Parameters	• **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.

Return values
- **None:**

LL_RTC_TIME_Init

Function name **ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)**

Function description Set the RTC current time.

Parameters
- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Time register is configured
  - ERROR: RTC Time register is not configured

Notes
- The user should call LL_RTC_TIME_StructInit() or the structure of time need to be initialized before time init()

LL_RTC_TIME_StructInit

Function name **void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)**

Function description Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters
- **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.

Return values
- **None:**

LL_RTC_ALARM_Init

Function name **ErrorStatus LL_RTC_ALARM_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**

Function description Set the RTC Alarm.

Parameters
- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARM registers are configured
  - ERROR: ALARM registers are not configured

Notes
- the user should call LL_RTC_ALARM_StructInit() or the structure of Alarm need to be initialized before Alarm init()

LL_RTC_ALARM_StructInit

Function name **void LL_RTC_ALARM_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)**

Function description Set each LL_RTC_AlarmTypeDef of ALARM field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values
- **None:**

LL_RTC_EnterInitMode

Function name **ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)**

Function description Enters the RTC Initialization mode.

Parameters
- **RTCx:** RTC Instance

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC is in Init mode
  – ERROR: RTC is not in Init mode

LL_RTC_ExitInitMode

Function name **ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)**

Function description Exit the RTC Initialization mode.

Parameters
- **RTCx:** RTC Instance

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC exited from in Init mode
  – ERROR: Not applicable

Notes
- When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.

LL_RTC_WaitForSynchro

Function name **ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)**

Function description Waits until the RTC registers are synchronized with RTC APB clock.

Parameters
- **RTCx:** RTC Instance

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC registers are synchronised
  – ERROR: RTC registers are not synchronised

Notes
- The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_TIME_SetCounter

Function name	**ErrorStatus LL_RTC_TIME_SetCounter (RTC_TypeDef * RTCx, uint32_t TimeCounter)**
Function description	Set the Time Counter.
Parameters	• **RTCx:** RTC Instance • **TimeCounter:** this value can be from 0 to 0xFFFFFFFF
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC Counter register configured – ERROR: Not applicable

LL_RTC_ALARM_SetCounter

Function name	**ErrorStatus LL_RTC_ALARM_SetCounter (RTC_TypeDef * RTCx, uint32_t AlarmCounter)**
Function description	Set Alarm Counter.
Parameters	• **RTCx:** RTC Instance • **AlarmCounter:** this value can be from 0 to 0xFFFFFFFF
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: RTC exited from in Init mode – ERROR: Not applicable

59.3 RTC Firmware driver defines

59.3.1 RTC

BACKUP

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17
LL_RTC_BKP_DR18
LL_RTC_BKP_DR19
LL_RTC_BKP_DR20
LL_RTC_BKP_DR21
LL_RTC_BKP_DR22
LL_RTC_BKP_DR23
LL_RTC_BKP_DR24
LL_RTC_BKP_DR25
LL_RTC_BKP_DR26
LL_RTC_BKP_DR27
LL_RTC_BKP_DR28
LL_RTC_BKP_DR29
LL_RTC_BKP_DR30
LL_RTC_BKP_DR31
LL_RTC_BKP_DR32
LL_RTC_BKP_DR33
LL_RTC_BKP_DR34
LL_RTC_BKP_DR35
LL_RTC_BKP_DR36
LL_RTC_BKP_DR37
LL_RTC_BKP_DR38
LL_RTC_BKP_DR39
LL_RTC_BKP_DR40
LL_RTC_BKP_DR41
LL_RTC_BKP_DR42

FORMAT

LL_RTC_FORMAT_BIN	Binary data format
LL_RTC_FORMAT_BCD	BCD data format

Clock Source to output on the Tamper Pin

LL_RTC_CALIB_OUTPUT_NONE	Calibration output disabled
LL_RTC_CALIB_OUTPUT_RTCCLOCK	Calibration output is RTC Clock with a frequency divided by 64 on the TAMPER Pin
LL_RTC_CALIB_OUTPUT_ALARM	Calibration output is Alarm pulse signal on the TAMPER pin
LL_RTC_CALIB_OUTPUT_SECOND	Calibration output is Second pulse signal on the TAMPER pin

Tamper Active Level

LL_RTC_TAMPER_ACTIVELEVEL_LOW	A high level on the TAMPER pin resets all data backup registers (if TPE bit is set)
LL_RTC_TAMPER_ACTIVELEVEL_HIGH	A low level on the TAMPER pin resets all data backup registers (if TPE bit is set)

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

60 LL SPI Generic Driver

60.1 SPI Firmware driver registers structures

60.1.1 LL_SPI_InitTypeDef

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***
  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***
  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***
  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_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_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***
  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.
  Note: The communication clock is derived from the master clock. The slave clock does not need to be set. This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.

- ***uint32_t LL_SPI_InitTypeDef::BitOrder***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

60.2 SPI Firmware driver API description

60.2.1 Detailed description of functions

LL_SPI_Enable

Function name	**__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)**
Function description	Enable SPI peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name	**__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)**
Function description	Disable SPI peripheral.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• When disabling the SPI, follow the procedure described in the Reference Manual.
Reference Manual to LL API cross reference:	• CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)**
Function description	Check if SPI peripheral is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR1 SPE LL_SPI_IsEnabled

reference:

LL_SPI_SetMode

Function name
__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description
Set SPI operation mode to Master or Slave.

Parameters
- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Return values
- **None:**

Notes
- This bit should not be changed when communication is ongoing.

Reference Manual to LL API cross reference:
- CR1 MSTR LL_SPI_SetMode
- CR1 SSI LL_SPI_SetMode

LL_SPI_GetMode

Function name
__STATIC_INLINE uint32_t LL_SPI_GetMode (SPI_TypeDef * SPIx)

Function description
Get SPI operation mode (Master or Slave)

Parameters
- **SPIx:** SPI Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Reference Manual to LL API cross reference:
- CR1 MSTR LL_SPI_GetMode
- CR1 SSI LL_SPI_GetMode

LL_SPI_SetClockPhase

Function name
__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)

Function description
Set clock phase.

Parameters
- **SPIx:** SPI Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Return values
- **None:**

Notes
- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross
- CR1 CPHA LL_SPI_SetClockPhase

reference:

LL_SPI_GetClockPhase

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (SPI_TypeDef * SPIx)**
Function description	Get clock phase.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_PHASE_1EDGE – LL_SPI_PHASE_2EDGE
Reference Manual to LL API cross reference:	• CR1 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name	**__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)**
Function description	Set clock polarity.
Parameters	• **SPIx:** SPI Instance • **ClockPolarity:** This parameter can be one of the following values: – LL_SPI_POLARITY_LOW – LL_SPI_POLARITY_HIGH
Return values	• **None:**
Notes	• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.
Reference Manual to LL API cross reference:	• CR1 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (SPI_TypeDef * SPIx)**
Function description	Get clock polarity.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_POLARITY_LOW – LL_SPI_POLARITY_HIGH
Reference Manual to LL API cross reference:	• CR1 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetBaudRatePrescaler

Function name
__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t BaudRate)

Function description
Set baud rate prescaler.

Parameters
- **SPIx:** SPI Instance
- **BaudRate:** This parameter can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Return values
- **None:**

Notes
- These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.

Reference Manual to LL API cross reference:
- CR1 BR LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name
__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (SPI_TypeDef * SPIx)

Function description
Get baud rate prescaler.

Parameters
- **SPIx:** SPI Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Reference Manual to LL API cross reference:
- CR1 BR LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name
__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description
Set transfer bit order.

Parameters
- **SPIx:** SPI Instance

- **BitOrder:** This parameter can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Return values
- **None:**

Notes
- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:
- CR1 LSBFIRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name **__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (SPI_TypeDef * SPIx)**

Function description Get transfer bit order.

Parameters
- **SPIx:** SPI Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Reference Manual to LL API cross reference:
- CR1 LSBFIRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name **__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)**

Function description Set transfer direction mode.

Parameters
- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values
- **None:**

Notes
- For Half-Duplex mode, Rx Direction is set by default. In master mode, the MOSI pin is used and in slave mode, the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:
- CR1 RXONLY LL_SPI_SetTransferDirection
- CR1 BIDIMODE LL_SPI_SetTransferDirection
- CR1 BIDIOE LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name **__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (SPI_TypeDef * SPIx)**

Function description	Get transfer direction mode.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_FULL_DUPLEX – LL_SPI_SIMPLEX_RX – LL_SPI_HALF_DUPLEX_RX – LL_SPI_HALF_DUPLEX_TX
Reference Manual to LL API cross reference:	• CR1 RXONLY LL_SPI_GetTransferDirection • CR1 BIDIMODE LL_SPI_GetTransferDirection • CR1 BIDIOE LL_SPI_GetTransferDirection

LL_SPI_SetDataWidth

Function name	**__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)**
Function description	Set frame data width.
Parameters	• **SPIx:** SPI Instance • **DataWidth:** This parameter can be one of the following values: – LL_SPI_DATAWIDTH_8BIT – LL_SPI_DATAWIDTH_16BIT
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 DFF LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (SPI_TypeDef * SPIx)**
Function description	Get frame data width.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value can be one of the following values: – LL_SPI_DATAWIDTH_8BIT – LL_SPI_DATAWIDTH_16BIT
Reference Manual to LL API cross reference:	• CR1 DFF LL_SPI_GetDataWidth

LL_SPI_EnableCRC

Function name	**__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)**
Function description	Enable CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**

Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name	**__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)**
Function description	Disable CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)**
Function description	Check if CRC is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.
Reference Manual to LL API cross reference:	• CR1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCNext

Function name	**__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)**
Function description	Set CRCNext to transfer CRC on the line.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit has to be written as soon as the last data is written in the SPIx_DR register.
Reference Manual to LL API cross reference:	• CR1 CRCNEXT LL_SPI_SetCRCNext

LL_SPI_SetCRCPolynomial

Function name	**__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)**
Function description	Set polynomial for CRC calculation.
Parameters	• **SPIx:** SPI Instance • **CRCPoly:** This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• CRCPR CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (SPI_TypeDef * SPIx)**
Function description	Get polynomial for CRC calculation.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• CRCPR CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_GetRxCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)**
Function description	Get Rx CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to LL API cross reference:	• RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name	**__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)**
Function description	Get Tx CRC.
Parameters	• **SPIx:** SPI Instance
Return values	• **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF
Reference Manual to	• TXCRCR TXCRC LL_SPI_GetTxCRC

LL API cross reference:

LL_SPI_SetNSSMode

Function name **__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)**

Function description Set NSS mode.

Parameters
- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Return values
- **None:**

Notes
- LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.

Reference Manual to LL API cross reference:
- CR1 SSM LL_SPI_SetNSSMode
-
- CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name **__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)**

Function description Get NSS mode.

Parameters
- **SPIx:** SPI Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:
- CR1 SSM LL_SPI_GetNSSMode
-
- CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_IsActiveFlag_RXNE

Function name **__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)**

Function description Check if Rx buffer is not empty.

Parameters
- **SPIx:** SPI Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- SR RXNE LL_SPI_IsActiveFlag_RXNE

LL_SPI_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer is empty.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TXE LL_SPI_IsActiveFlag_TXE

LL_SPI_IsActiveFlag_CRCERR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (SPI_TypeDef * SPIx)**
Function description	Get CRC error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CRCERR LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (SPI_TypeDef * SPIx)**
Function description	Get mode fault error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Get overrun error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_BSY

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_BSY (SPI_TypeDef * SPIx)**
Function description	Get busy flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Notes	• The BSY flag is cleared under any one of the following conditions: -When the SPI is correctly disabled -When a fault is detected in Master mode (MODF bit set to 1) -In Master mode, when it finishes a data transmission and no new data is ready to be sent -In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.
Reference Manual to LL API cross reference:	• SR BSY LL_SPI_IsActiveFlag_BSY

LL_SPI_ClearFlag_CRCERR

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)**
Function description	Clear CRC error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CRCERR LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)**
Function description	Clear mode fault error flag.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the SPIx_SR register followed by a write access to the SPIx_CR1 register
Reference Manual to LL API cross reference:	• SR MODF LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_OVR

Function name	**__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)**
Function description	Clear overrun error flag.

Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the SPIx_DR register followed by a read access to the SPIx_SR register
Reference Manual to LL API cross reference:	• SR OVR LL_SPI_ClearFlag_OVR

LL_SPI_EnableIT_ERR

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Enable error interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_EnableIT_ERR

LL_SPI_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Enable Rx buffer not empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_EnableIT_RXNE

LL_SPI_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Enable Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_EnableIT_TXE

LL_SPI_DisableIT_ERR

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef * SPIx)**
Function description	Disable error interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Notes	• This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_SPI_DisableIT_ERR

LL_SPI_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Disable Rx buffer not empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_DisableIT_RXNE

LL_SPI_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef * SPIx)**
Function description	Disable Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_DisableIT_TXE

LL_SPI_IsEnabledIT_ERR

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (SPI_TypeDef * SPIx)**
Function description	Check if error interrupt is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross	• CR2 ERRIE LL_SPI_IsEnabledIT_ERR

reference:

LL_SPI_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)**
Function description	Check if Rx buffer not empty interrupt is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_SPI_IsEnabledIT_RXNE

LL_SPI_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXE (SPI_TypeDef * SPIx)**
Function description	Check if Tx buffer empty interrupt.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_SPI_IsEnabledIT_TXE

LL_SPI_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Rx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Rx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Enable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Disable DMA Tx.
Parameters	• **SPIx:** SPI Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)**
Function description	Check if DMA Tx is enabled.
Parameters	• **SPIx:** SPI Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (SPI_TypeDef * SPIx)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **SPIx:** SPI Instance
Return values	• **Address:** of data register
Reference Manual to LL API cross reference:	• DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)**
Function description	Read 8-Bits in the data register.
Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name	**__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)**
Function description	Read 16-Bits in the data register.
Parameters	• **SPIx:** SPI Instance
Return values	• **RxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Reference Manual to LL API cross reference:	• DR DR LL_SPI_ReceiveData16

LL_SPI_TransmitData8

Function name	**__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)**
Function description	Write 8-Bits in the data register.
Parameters	• **SPIx:** SPI Instance • **TxData:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross	• DR DR LL_SPI_TransmitData8

reference:

LL_SPI_TransmitData16

Function name	**__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)**
Function description	Write 16-Bits in the data register.
Parameters	• **SPIx:** SPI Instance • **TxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_SPI_TransmitData16

LL_SPI_DeInit

Function name	**ErrorStatus LL_SPI_DeInit (SPI_TypeDef * SPIx)**
Function description	De-initialize the SPI registers to their default reset values.
Parameters	• **SPIx:** SPI Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: SPI registers are de-initialized – ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name	**ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)**
Function description	Initialize the SPI registers according to the specified parameters in SPI_InitStruct.
Parameters	• **SPIx:** SPI Instance • **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure
Return values	• **An:** ErrorStatus enumeration value. (Return always SUCCESS)
Notes	• As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name	**void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)**
Function description	Set each LL_SPI_InitTypeDef field to default value.
Parameters	• **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.
Return values	• **None:**

60.3 SPI Firmware driver defines

60.3.1 SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_DIV2	BaudRate control equal to fPCLK/2
LL_SPI_BAUDRATEPRESCALER_DIV4	BaudRate control equal to fPCLK/4
LL_SPI_BAUDRATEPRESCALER_DIV8	BaudRate control equal to fPCLK/8
LL_SPI_BAUDRATEPRESCALER_DIV16	BaudRate control equal to fPCLK/16
LL_SPI_BAUDRATEPRESCALER_DIV32	BaudRate control equal to fPCLK/32
LL_SPI_BAUDRATEPRESCALER_DIV64	BaudRate control equal to fPCLK/64
LL_SPI_BAUDRATEPRESCALER_DIV128	BaudRate control equal to fPCLK/128
LL_SPI_BAUDRATEPRESCALER_DIV256	BaudRate control equal to fPCLK/256

Transmission Bit Order

LL_SPI_LSB_FIRST	Data is transmitted/received with the LSB first
LL_SPI_MSB_FIRST	Data is transmitted/received with the MSB first

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE	CRC calculation disabled
LL_SPI_CRCCALCULATION_ENABLE	CRC calculation enabled

Datawidth

LL_SPI_DATAWIDTH_8BIT	Data length for SPI transfer: 8 bits
LL_SPI_DATAWIDTH_16BIT	Data length for SPI transfer: 16 bits

Get Flags Defines

LL_SPI_SR_RXNE	Rx buffer not empty flag
LL_SPI_SR_TXE	Tx buffer empty flag
LL_SPI_SR_BSY	Busy flag
LL_SPI_SR_CRCERR	CRC error flag
LL_SPI_SR_MODF	Mode fault flag
LL_SPI_SR_OVR	Overrun flag
LL_SPI_SR_FRE	TI mode frame format error flag

IT Defines

LL_I2S_CR2_RXNEIE	Rx buffer not empty interrupt enable
LL_I2S_CR2_TXEIE	Tx buffer empty interrupt enable
LL_I2S_CR2_ERRIE	Error interrupt enable
LL_SPI_CR2_RXNEIE	Rx buffer not empty interrupt enable
LL_SPI_CR2_TXEIE	Tx buffer empty interrupt enable
LL_SPI_CR2_ERRIE	Error interrupt enable

Operation Mode

LL_SPI_MODE_MASTER Master configuration

LL_SPI_MODE_SLAVE Slave configuration

Slave Select Pin Mode

LL_SPI_NSS_SOFT NSS managed internally. NSS pin not used and free

LL_SPI_NSS_HARD_INPUT NSS pin used in Input. Only used in Master mode

LL_SPI_NSS_HARD_OUTPUT NSS pin used in Output. Only used in Slave mode as chip select

Clock Phase

LL_SPI_PHASE_1EDGE First clock transition is the first data capture edge

LL_SPI_PHASE_2EDGE Second clock transition is the first data capture edge

Clock Polarity

LL_SPI_POLARITY_LOW Clock to 0 when idle

LL_SPI_POLARITY_HIGH Clock to 1 when idle

Transfer Mode

LL_SPI_FULL_DUPLEX Full-Duplex mode. Rx and Tx transfer on 2 lines

LL_SPI_SIMPLEX_RX Simplex Rx mode. Rx transfer only on 1 line

LL_SPI_HALF_DUPLEX_RX Half-Duplex Rx mode. Rx transfer on 1 line

LL_SPI_HALF_DUPLEX_TX Half-Duplex Tx mode. Tx transfer on 1 line

Common Write and read registers Macros

LL_SPI_WriteReg

Description:

- Write a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SPI_ReadReg

Description:

- Read a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:

- Register: value

61 LL SYSTEM Generic Driver

61.1 SYSTEM Firmware driver API description

61.1.1 Detailed description of functions

LL_DBGMCU_GetDeviceID

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)**
Function description	Return the device identifier.
Return values	• **Values:** between Min_Data=0x00 and Max_Data=0xFFF
Notes	• For Low Density devices, the device ID is 0x412 • For Medium Density devices, the device ID is 0x410 • For High Density devices, the device ID is 0x414 • For XL Density devices, the device ID is 0x430 • For Connectivity Line devices, the device ID is 0x418
Reference Manual to LL API cross reference:	• DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)**
Function description	Return the device revision identifier.
Return values	• **Values:** between Min_Data=0x00 and Max_Data=0xFFFF
Notes	• This field indicates the revision of the device. For example, it is read as revA -> 0x1000,for Low Density devices For example, it is read as revA -> 0x0000, revB -> 0x2000, revZ -> 0x2001, rev1,2,3,X or Y -> 0x2003,for Medium Density devices For example, it is read as revA or 1 -> 0x1000, revZ -> 0x1001,rev1,2,3,X or Y -> 0x1003,for Medium Density devices For example, it is read as revA or 1 -> 0x1003,for XL Density devices For example, it is read as revA -> 0x1000, revZ -> 0x1001 for Connectivity line devices
Reference Manual to LL API cross reference:	• DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGSleepMode

Function name	**__STATIC_INLINE void LL_DBGMCU_EnableDBGSleepMode (void)**
Function description	Enable the Debug Module during SLEEP mode.
Return values	• **None:**
Reference Manual to	• DBGMCU_CR DBG_SLEEP

LL API cross reference: LL_DBGMCU_EnableDBGSleepMode

LL_DBGMCU_DisableDBGSleepMode

Function name **__STATIC_INLINE void LL_DBGMCU_DisableDBGSleepMode (void)**

Function description Disable the Debug Module during SLEEP mode.

Return values
- **None:**

Reference Manual to LL API cross reference:
- DBGMCU_CR DBG_SLEEP LL_DBGMCU_DisableDBGSleepMode

LL_DBGMCU_EnableDBGStopMode

Function name **__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)**

Function description Enable the Debug Module during STOP mode.

Return values
- **None:**

Reference Manual to LL API cross reference:
- DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode

LL_DBGMCU_DisableDBGStopMode

Function name **__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)**

Function description Disable the Debug Module during STOP mode.

Return values
- **None:**

Reference Manual to LL API cross reference:
- DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode

LL_DBGMCU_EnableDBGStandbyMode

Function name **__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)**

Function description Enable the Debug Module during STANDBY mode.

Return values
- **None:**

Reference Manual to LL API cross reference:
- DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name **__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)**

Function description	Disable the Debug Module during STANDBY mode.
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

LL_DBGMCU_SetTracePinAssignment

Function name	**__STATIC_INLINE void LL_DBGMCU_SetTracePinAssignment (uint32_t PinAssignment)**
Function description	Set Trace pin assignment control.
Parameters	• **PinAssignment:** This parameter can be one of the following values: – LL_DBGMCU_TRACE_NONE – LL_DBGMCU_TRACE_ASYNCH – LL_DBGMCU_TRACE_SYNCH_SIZE1 – LL_DBGMCU_TRACE_SYNCH_SIZE2 – LL_DBGMCU_TRACE_SYNCH_SIZE4
Return values	• **None:**
Reference Manual to LL API cross reference:	• DBGMCU_CR TRACE_IOEN LL_DBGMCU_SetTracePinAssignment • DBGMCU_CR TRACE_MODE LL_DBGMCU_SetTracePinAssignment

LL_DBGMCU_GetTracePinAssignment

Function name	**__STATIC_INLINE uint32_t LL_DBGMCU_GetTracePinAssignment (void)**
Function description	Get Trace pin assignment control.
Return values	• **Returned:** value can be one of the following values: – LL_DBGMCU_TRACE_NONE – LL_DBGMCU_TRACE_ASYNCH – LL_DBGMCU_TRACE_SYNCH_SIZE1 – LL_DBGMCU_TRACE_SYNCH_SIZE2 – LL_DBGMCU_TRACE_SYNCH_SIZE4
Reference Manual to LL API cross reference:	• DBGMCU_CR TRACE_IOEN LL_DBGMCU_GetTracePinAssignment • DBGMCU_CR TRACE_MODE LL_DBGMCU_GetTracePinAssignment

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name	**__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)**
Function description	Freeze APB1 peripherals (group1 peripherals)
Parameters	• **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices. – LL_DBGMCU_APB1_GRP1_TIM2_STOP

  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_TIM12_STOP
  - LL_DBGMCU_APB1_GRP1_TIM13_STOP
  - LL_DBGMCU_APB1_GRP1_TIM14_STOP
  - LL_DBGMCU_APB1_GRP1_RTC_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_CAN1_STOP (*)
  - LL_DBGMCU_APB1_GRP1_CAN2_STOP (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR_APB1 DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM4_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM5_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM6_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM7_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM12_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM13_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_TIM14_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_I2C1_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_I2C2_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_CAN1_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_CR_APB1 DBG_CAN2_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name **__STATIC_INLINE void**

LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_TIM12_STOP
  - LL_DBGMCU_APB1_GRP1_TIM13_STOP
  - LL_DBGMCU_APB1_GRP1_TIM14_STOP
  - LL_DBGMCU_APB1_GRP1_RTC_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_CAN1_STOP (*)
  - LL_DBGMCU_APB1_GRP1_CAN2_STOP (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR_APB1 DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM4_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM5_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM6_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM7_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM12_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM13_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_TIM14_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_I2C1_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_I2C2_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_CAN1_STOP

LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBGMCU_CR_APB1 DBG_CAN2_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name
__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)

Function description
Freeze APB2 peripherals.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM9_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM10_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM11_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- DBGMCU_CR_APB2 DBG_TIM1_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM8_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM9_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM10_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM11_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM15_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM16_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM17_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name
__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description
Unfreeze APB2 peripherals.

Parameters
- **Periphs:** This parameter can be a combination of the following values: (*) value not defined in all devices.
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM9_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM10_STOP (*)
  - LL_DBGMCU_APB2_GRP1_TIM11_STOP (*)

- LL_DBGMCU_APB2_GRP1_TIM15_STOP (*)
- LL_DBGMCU_APB2_GRP1_TIM16_STOP (*)
- LL_DBGMCU_APB2_GRP1_TIM17_STOP (*)

Return values
- **None:**

Reference Manual to LL API cross reference:
- DBGMCU_CR_APB2 DBG_TIM1_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM8_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM9_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM10_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM11_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM15_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM16_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_CR_APB2 DBG_TIM17_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_FLASH_SetLatency

Function name: **__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)**

Function description: Set FLASH Latency.

Parameters
- **Latency:** This parameter can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2

Return values
- **None:**

Reference Manual to LL API cross reference:
- FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name: **__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)**

Function description: Get FLASH Latency.

Return values
- **Returned:** value can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2

Reference Manual to LL API cross reference:
- FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_FLASH_EnablePrefetch

Function name	**__STATIC_INLINE void LL_FLASH_EnablePrefetch (void)**
Function description	Enable Prefetch.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTBE LL_FLASH_EnablePrefetch

LL_FLASH_DisablePrefetch

Function name	**__STATIC_INLINE void LL_FLASH_DisablePrefetch (void)**
Function description	Disable Prefetch.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTBE LL_FLASH_DisablePrefetch

LL_FLASH_IsPrefetchEnabled

Function name	**__STATIC_INLINE uint32_t LL_FLASH_IsPrefetchEnabled (void)**
Function description	Check if Prefetch buffer is enabled.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• FLASH_ACR PRFTBS LL_FLASH_IsPrefetchEnabled

LL_FLASH_EnableHalfCycleAccess

Function name	**__STATIC_INLINE void LL_FLASH_EnableHalfCycleAccess (void)**
Function description	Enable Flash Half Cycle Access.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR HLFCYA LL_FLASH_EnableHalfCycleAccess

LL_FLASH_DisableHalfCycleAccess

Function name	**__STATIC_INLINE void LL_FLASH_DisableHalfCycleAccess (void)**
Function description	Disable Flash Half Cycle Access.
Return values	• **None:**
Reference Manual to LL API cross reference:	• FLASH_ACR HLFCYA LL_FLASH_DisableHalfCycleAccess

LL_FLASH_IsHalfCycleAccessEnabled

Function name	**__STATIC_INLINE uint32_t LL_FLASH_IsHalfCycleAccessEnabled (void)**
Function description	Check if Flash Half Cycle Access is enabled or not.
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• FLASH_ACR HLFCYA LL_FLASH_IsHalfCycleAccessEnabled

61.2 SYSTEM Firmware driver defines

61.2.1 SYSTEM

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP	TIM2 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM3_STOP	TIM3 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM4_STOP	TIM4 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM5_STOP	TIM5 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM6_STOP	TIM6 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM7_STOP	TIM7 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM12_STOP	TIM12 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM13_STOP	TIM13 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_TIM14_STOP	TIM14 counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_RTC_STOP	RTC counter stopped when core is halted
LL_DBGMCU_APB1_GRP1_WWDG_STOP	Debug Window Watchdog stopped when Core is halted
LL_DBGMCU_APB1_GRP1_IWDG_STOP	Debug Independent Watchdog stopped when Core is halted
LL_DBGMCU_APB1_GRP1_I2C1_STOP	I2C1 SMBUS timeout mode stopped when Core is halted
LL_DBGMCU_APB1_GRP1_I2C2_STOP	I2C2 SMBUS timeout mode stopped when Core is halted
LL_DBGMCU_APB1_GRP1_CAN1_STOP	CAN1 debug stopped when Core is halted

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP	TIM1 counter stopped when core is halted
LL_DBGMCU_APB2_GRP1_TIM8_STOP	TIM8 counter stopped when core is halted
LL_DBGMCU_APB2_GRP1_TIM9_STOP	TIM9 counter stopped when core is halted
LL_DBGMCU_APB2_GRP1_TIM10_STOP	TIM10 counter stopped when core is halted
LL_DBGMCU_APB2_GRP1_TIM11_STOP	TIM11 counter stopped when core is halted

FLASH LATENCY

LL_FLASH_LATENCY_0	FLASH Zero Latency cycle
LL_FLASH_LATENCY_1	FLASH One Latency cycle
LL_FLASH_LATENCY_2	FLASH Two wait states

DBGMCU TRACE Pin Assignment

LL_DBGMCU_TRACE_NONE	TRACE pins not assigned (default state)
LL_DBGMCU_TRACE_ASYNCH	TRACE pin assignment for Asynchronous Mode
LL_DBGMCU_TRACE_SYNCH_SIZE1	TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1
LL_DBGMCU_TRACE_SYNCH_SIZE2	TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2
LL_DBGMCU_TRACE_SYNCH_SIZE4	TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

62 LL TIM Generic Driver

62.1 TIM Firmware driver registers structures

62.1.1 LL_TIM_InitTypeDef

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint8_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_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.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***
  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint8_t LL_TIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:the number of PWM periods in edge-aligned modethe number of half PWM period in center-aligned mode This parameter must be a number between 0x00 and 0xFF. This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

62.1.2 LL_TIM_OC_InitTypeDef

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***
  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***
  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***
  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***
  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).
- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

62.1.3 LL_TIM_IC_InitTypeDef

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***
  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of

TIM_LL_EC_ICPSC.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

62.1.4 LL_TIM_ENCODER_InitTypeDef

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***
  Specifies the encoder resolution (x2 or x4). This parameter can be a value of *TIM_LL_EC_ENCODERMODE*.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of *TIM_LL_EC_IC_POLARITY*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***
  Specifies the TI1 input source This parameter can be a value of *TIM_LL_EC_ACTIVEINPUT*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***
  Specifies the active edge of TI2 input. This parameter can be a value of *TIM_LL_EC_IC_POLARITY*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***
  Specifies the TI2 input source This parameter can be a value of *TIM_LL_EC_ACTIVEINPUT*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***
  Specifies the TI2 input prescaler value. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***
  Specifies the TI2 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

62.1.5 LL_TIM_HALLSENSOR_InitTypeDef

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***
  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

62.1.6 LL_TIM_BDTR_InitTypeDef

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***
  Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_LL_EC_OSSR***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note: This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***
  Specifies the Off-State used in Idle state. This parameter can be a value of

*TIM_LL_EC_OSSI*This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
Note: This bit-field cannot be modified as long as LOCK level 2 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***
  Specifies the LOCK level parameters. This parameter can be a value of *TIM_LL_EC_LOCKLEVEL*
  Note: The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.
- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***
  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**
  Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.
- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***
  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of *TIM_LL_EC_BREAK_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***
  Specifies the TIM Break Input pin polarity. This parameter can be a value of *TIM_LL_EC_BREAK_POLARITY*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***
  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of *TIM_LL_EC_AUTOMATICOUTPUT_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**
  Note: This bit-field can not be modified as long as LOCK level 1 has been programmed.

62.2 TIM Firmware driver API description

62.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name	**__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)**
Function description	Enable timer counter.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name	**__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)**
Function description	Disable timer counter.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (TIM_TypeDef * TIMx)**
Function description	Indicates whether the timer counter is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name	**__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Enable update event generation.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name	**__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Disable update event generation.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (TIM_TypeDef * TIMx)**
Function description	Indicates whether update event generation is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name	**__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)**
Function description	Set update event source.
Parameters	• **TIMx:** Timer instance • **UpdateSource:** This parameter can be one of the following values: – LL_TIM_UPDATESOURCE_REGULAR – LL_TIM_UPDATESOURCE_COUNTER
Return values	• **None:**
Notes	• Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller • Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.
Reference Manual to LL API cross reference:	• CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (TIM_TypeDef * TIMx)**
Function description	Get actual event update source.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_UPDATESOURCE_REGULAR – LL_TIM_UPDATESOURCE_COUNTER
Reference Manual to LL API cross reference:	• CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name	**__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)**
Function description	Set one pulse mode (one shot v.s.
Parameters	• **TIMx:** Timer instance • **OnePulseMode:** This parameter can be one of the following values: – LL_TIM_ONEPULSEMODE_SINGLE – LL_TIM_ONEPULSEMODE_REPETITIVE
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (TIM_TypeDef * TIMx)**
Function description	Get actual one pulse mode.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_ONEPULSEMODE_SINGLE – LL_TIM_ONEPULSEMODE_REPETITIVE
Reference Manual to LL API cross reference:	• CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name	**__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)**
Function description	Set the timer counter counting mode.
Parameters	• **TIMx:** Timer instance • **CounterMode:** This parameter can be one of the following values: – LL_TIM_COUNTERMODE_UP – LL_TIM_COUNTERMODE_DOWN – LL_TIM_COUNTERMODE_CENTER_UP – LL_TIM_COUNTERMODE_CENTER_DOWN – LL_TIM_COUNTERMODE_CENTER_UP_DOWN
Return values	• **None:**
Notes	• Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
Reference Manual to LL API cross	• CR1 DIR LL_TIM_SetCounterMode

reference:
- CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name: **__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (TIM_TypeDef * TIMx)**

Function description: Get actual counter mode.

Parameters:
- **TIMx:** Timer instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Notes:
- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.

Reference Manual to LL API cross reference:
- CR1 DIR LL_TIM_GetCounterMode
- CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name: **__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)**

Function description: Enable auto-reload (ARR) preload.

Parameters:
- **TIMx:** Timer instance

Return values:
- **None:**

Reference Manual to LL API cross reference:
- CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name: **__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)**

Function description: Disable auto-reload (ARR) preload.

Parameters:
- **TIMx:** Timer instance

Return values:
- **None:**

Reference Manual to LL API cross reference:
- CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name: **__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload**

(TIM_TypeDef * TIMx)

Function description: Indicates whether auto-reload (ARR) preload is enabled.

Parameters:
- **TIMx:** Timer instance

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name: **__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)**

Function description: Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters:
- **TIMx:** Timer instance
- **ClockDivision:** This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Return values:
- **None:**

Notes:
- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:
- CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name: **__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (TIM_TypeDef * TIMx)**

Function description: Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters:
- **TIMx:** Timer instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Notes:
- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross
- CR1 CKD LL_TIM_GetClockDivision

reference:

LL_TIM_SetCounter

Function name	**__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)**
Function description	Set the counter value.
Parameters	• **TIMx:** Timer instance • **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF)
Return values	• **None:**
Reference Manual to LL API cross reference:	• CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetCounter (TIM_TypeDef * TIMx)**
Function description	Get the counter value.
Parameters	• **TIMx:** Timer instance
Return values	• **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF)
Reference Manual to LL API cross reference:	• CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetDirection (TIM_TypeDef * TIMx)**
Function description	Get the current direction of the counter.
Parameters	• **TIMx:** Timer instance
Return values	• **Returned:** value can be one of the following values: – LL_TIM_COUNTERDIRECTION_UP – LL_TIM_COUNTERDIRECTION_DOWN
Reference Manual to LL API cross reference:	• CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name	**__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)**
Function description	Set the prescaler value.
Parameters	• **TIMx:** Timer instance

- **Prescaler:** between Min_Data=0 and Max_Data=65535

Return values
- **None:**

Notes
- The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
- The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.
- Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter

Reference Manual to LL API cross reference:
- PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name **__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (TIM_TypeDef * TIMx)**

Function description Get the prescaler value.

Parameters
- **TIMx:** Timer instance

Return values
- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:
- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name **__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)**

Function description Set the auto-reload value.

Parameters
- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values
- **None:**

Notes
- The counter is blocked while the auto-reload value is null.
- Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter

Reference Manual to LL API cross reference:
- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name **__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (TIM_TypeDef * TIMx)**

Function description Get the auto-reload value.

Parameters
- **TIMx:** Timer instance

Return values	• **Auto-reload:** value
Reference Manual to LL API cross reference:	• ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name	**__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)**
Function description	Set the repetition counter value.
Parameters	• **TIMx:** Timer instance • **RepetitionCounter:** between Min_Data=0 and Max_Data=255
Return values	• **None:**
Notes	• Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.
Reference Manual to LL API cross reference:	• RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name	**__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (TIM_TypeDef * TIMx)**
Function description	Get the repetition counter value.
Parameters	• **TIMx:** Timer instance
Return values	• **Repetition:** counter value
Notes	• Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.
Reference Manual to LL API cross reference:	• RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_CC_EnablePreload

Function name	**__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)**
Function description	Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs.

- Only on channels that have a complementary output.
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:
- CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name **__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)**

Function description Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters
- **TIMx:** Timer instance

Return values
- **None:**

Notes
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:
- CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_SetUpdate

Function name **__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)**

Function description Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).

Parameters
- **TIMx:** Timer instance
- **CCUpdateSource:** This parameter can be one of the following values:
  - LL_TIM_CCUPDATESOURCE_COMG_ONLY
  - LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Return values
- **None:**

Notes
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:
- CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name **__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)**

Function description Set the trigger of the capture/compare DMA request.

Parameters
- **TIMx:** Timer instance
- **DMAReqTrigger:** This parameter can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name
__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (TIM_TypeDef * TIMx)

Function description
Get actual trigger of the capture/compare DMA request.

Parameters
- **TIMx:** Timer instance

Return values
- **Returned:** value can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Reference Manual to LL API cross reference:
- CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name
__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)

Function description
Set the lock level to freeze the configuration of several capture/compare parameters.

Parameters
- **TIMx:** Timer instance
- **LockLevel:** This parameter can be one of the following values:
  - LL_TIM_LOCKLEVEL_OFF
  - LL_TIM_LOCKLEVEL_1
  - LL_TIM_LOCKLEVEL_2
  - LL_TIM_LOCKLEVEL_3

Return values
- **None:**

Notes
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.

Reference Manual to LL API cross reference:
- BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name
__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description Enable capture/compare channels.

Parameters
- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCER CC1E LL_TIM_CC_EnableChannel
- CCER CC1NE LL_TIM_CC_EnableChannel
- CCER CC2E LL_TIM_CC_EnableChannel
- CCER CC2NE LL_TIM_CC_EnableChannel
- CCER CC3E LL_TIM_CC_EnableChannel
- CCER CC3NE LL_TIM_CC_EnableChannel
- CCER CC4E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name **__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)**

Function description Disable capture/compare channels.

Parameters
- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name **__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)**

Function description: Indicate whether channel(s) is(are) enabled.

Parameters:
- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name: **__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)**

Function description: Configure an output channel.

Parameters:
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
  - LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH

Return values:
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 CC1S LL_TIM_OC_ConfigOutput
- CCMR1 CC2S LL_TIM_OC_ConfigOutput
- CCMR2 CC3S LL_TIM_OC_ConfigOutput
- CCMR2 CC4S LL_TIM_OC_ConfigOutput
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P LL_TIM_OC_ConfigOutput
- CR2 OIS1 LL_TIM_OC_ConfigOutput
- CR2 OIS2 LL_TIM_OC_ConfigOutput

- CR2 OIS3 LL_TIM_OC_ConfigOutput
- CR2 OIS4 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name
__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)

Function description
Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Mode:** This parameter can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name
__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Get the output compare mode of an output channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **Returned:** value can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1

– LL_TIM_OCMODE_PWM2

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_GetMode
- CCMR1 OC2M LL_TIM_OC_GetMode
- CCMR2 OC3M LL_TIM_OC_GetMode
- CCMR2 OC4M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name **__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)**

Function description Set the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name **__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description Get the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH

– LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name **__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)**

Function description Set the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
- **IdleState:** This parameter can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 OIS1N LL_TIM_OC_SetIdleState
- CR2 OIS2 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name **__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description Get the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2

  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4

Return values
- **Returned:** value can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Reference Manual to LL API cross reference:
- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS1N LL_TIM_OC_GetIdleState
- CR2 OIS2 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS3 LL_TIM_OC_GetIdleState
- CR2 OIS3N LL_TIM_OC_GetIdleState
- CR2 OIS4 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name
__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Enable fast mode for the output channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **None:**

Notes
- Acts only if the channel is configured in PWM1 or PWM2 mode.

Reference Manual to LL API cross reference:
- CCMR1 OC1FE LL_TIM_OC_EnableFast
- CCMR1 OC2FE LL_TIM_OC_EnableFast
- CCMR2 OC3FE LL_TIM_OC_EnableFast
- CCMR2 OC4FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name
__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Disable fast mode for the output channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **None:**

Reference Manual to
- CCMR1 OC1FE LL_TIM_OC_DisableFast

LL API cross reference:
- CCMR1 OC2FE LL_TIM_OC_DisableFast
- CCMR2 OC3FE LL_TIM_OC_DisableFast
- CCMR2 OC4FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name: **__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description: Indicates whether fast mode is enabled for the output channel.

Parameters:
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCMR1 OC1FE LL_TIM_OC_IsEnabledFast
- CCMR1 OC2FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC3FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC4FE LL_TIM_OC_IsEnabledFast
-

LL_TIM_OC_EnablePreload

Function name: **__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description: Enable compare register (TIMx_CCRx) preload for the output channel.

Parameters:
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values:
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 OC1PE LL_TIM_OC_EnablePreload
- CCMR1 OC2PE LL_TIM_OC_EnablePreload
- CCMR2 OC3PE LL_TIM_OC_EnablePreload
- CCMR2 OC4PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name: **__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description: Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters:
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:

- LL_TIM_CHANNEL_CH1
- LL_TIM_CHANNEL_CH2
- LL_TIM_CHANNEL_CH3
- LL_TIM_CHANNEL_CH4

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 OC1PE LL_TIM_OC_DisablePreload
- CCMR1 OC2PE LL_TIM_OC_DisablePreload
- CCMR2 OC3PE LL_TIM_OC_DisablePreload
- CCMR2 OC4PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name
__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload
- CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload
-

LL_TIM_OC_EnableClear

Function name
__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Enable clearing the output channel on an external event.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **None:**

Notes
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross
- CCMR1 OC1CE LL_TIM_OC_EnableClear

reference:
- CCMR1 OC2CE LL_TIM_OC_EnableClear
- CCMR2 OC3CE LL_TIM_OC_EnableClear
- CCMR2 OC4CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name: **__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description: Disable clearing the output channel on an external event.

Parameters:
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values:
- **None:**

Notes:
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:
- CCMR1 OC1CE LL_TIM_OC_DisableClear
- CCMR1 OC2CE LL_TIM_OC_DisableClear
- CCMR2 OC3CE LL_TIM_OC_DisableClear
- CCMR2 OC4CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name: **__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description: Indicates clearing the output channel on an external event is enabled for the output channel.

Parameters:
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values:
- **State:** of bit (1 or 0).

Notes:
- This function enables clearing the output channel on an external event.
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:
- CCMR1 OC1CE LL_TIM_OC_IsEnabledClear
- CCMR1 OC2CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC3CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC4CE LL_TIM_OC_IsEnabledClear

-

LL_TIM_OC_SetDeadTime

Function name	**__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)**
Function description	Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge if the Ocx and OCxN signals).
Parameters	• **TIMx:** Timer instance • **DeadTime:** between Min_Data=0 and Max_Data=255
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance. • Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter
Reference Manual to LL API cross reference:	• BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 1 (TIMx_CCR1).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 2 (TIMx_CCR2).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**

Notes	• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 3 (TIMx_CCR3).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name	**__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)**
Function description	Set compare value for output channel 4 (TIMx_CCR4).
Parameters	• **TIMx:** Timer instance • **CompareValue:** between Min_Data=0 and Max_Data=65535
Return values	• **None:**
Notes	• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_GetCompareCH1

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR1) set for output channel 1.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and

Max_Data=65535)

Notes	• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR2) set for output channel 2.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR3) set for output channel 3.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name	**__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (TIM_TypeDef * TIMx)**
Function description	Get compare value (TIMx_CCR4) set for output channel 4.
Parameters	• **TIMx:** Timer instance
Return values	• **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.

Reference Manual to LL API cross reference:
- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_IC_Config

Function name
__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description
Configure input channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC
  - LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8
  - LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8
  - LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
-

LL_TIM_IC_SetActiveInput

Function name
__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description
Set the active input.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name
__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Get the current active input.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **Returned:** value can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Reference Manual to LL API cross reference:
- CCMR1 CC1S LL_TIM_IC_GetActiveInput
- CCMR1 CC2S LL_TIM_IC_GetActiveInput
- CCMR2 CC3S LL_TIM_IC_GetActiveInput
- CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name
__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)

Function description
Set the prescaler of input channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 IC1PSC LL_TIM_IC_SetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name **__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description Get the current prescaler value acting on an input channel.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **Returned:** value can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Reference Manual to LL API cross reference:
- CCMR1 IC1PSC LL_TIM_IC_GetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name **__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)**

Function description Set the input filter duration.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3

  - LL_TIM_CHANNEL_CH4
- **ICFilter:** This parameter can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Return values
- **None:**

Reference Manual to LL API cross reference:
- CCMR1 IC1F LL_TIM_IC_SetFilter
- CCMR1 IC2F LL_TIM_IC_SetFilter
- CCMR2 IC3F LL_TIM_IC_SetFilter
- CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name
__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (TIM_TypeDef * TIMx, uint32_t Channel)

Function description
Get the input filter duration.

Parameters
- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values
- **Returned:** value can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5

- LL_TIM_IC_FILTER_FDIV32_N6
- LL_TIM_IC_FILTER_FDIV32_N8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_GetFilter
- CCMR1 IC2F LL_TIM_IC_GetFilter
- CCMR2 IC3F LL_TIM_IC_GetFilter
- CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name **__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)**

Function description Set the input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPolarity:** This parameter can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_SetPolarity
- CCER CC1NP LL_TIM_IC_SetPolarity
- CCER CC2P LL_TIM_IC_SetPolarity
- CCER CC2NP LL_TIM_IC_SetPolarity
- CCER CC3P LL_TIM_IC_SetPolarity
- CCER CC3NP LL_TIM_IC_SetPolarity
- CCER CC4P LL_TIM_IC_SetPolarity
-

LL_TIM_IC_GetPolarity

Function name **__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)**

Function description Get the current input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING

Reference Manual to LL API cross

- CCER CC1P LL_TIM_IC_GetPolarity
- CCER CC1NP LL_TIM_IC_GetPolarity

reference:
- CCER CC2P LL_TIM_IC_GetPolarity
- CCER CC2NP LL_TIM_IC_GetPolarity
- CCER CC3P LL_TIM_IC_GetPolarity
- CCER CC3NP LL_TIM_IC_GetPolarity
- CCER CC4P LL_TIM_IC_GetPolarity
-

LL_TIM_IC_EnableXORCombination

Function name: **__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)**

Function description: Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).

Parameters:
- **TIMx:** Timer instance

Return values:
- **None:**

Notes:
- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:
- CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name: **__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)**

Function description: Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.

Parameters:
- **TIMx:** Timer instance

Return values:
- **None:**

Notes:
- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:
- CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name: **__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (TIM_TypeDef * TIMx)**

Function description: Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.

Parameters:
- **TIMx:** Timer instance

Return values:
- **State:** of bit (1 or 0).

Notes:
- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross
- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

reference:

LL_TIM_IC_GetCaptureCH1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 1.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 2.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 3.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (TIM_TypeDef * TIMx)**
Function description	Get captured value for input channel 4.
Parameters	• **TIMx:** Timer instance
Return values	• **CapturedValue:** (between Min_Data=0 and Max_Data=65535)
Notes	• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
Reference Manual to LL API cross reference:	• CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name	**__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)**
Function description	Enable external clock mode 2.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal. • Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name	**__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)**
Function description	Disable external clock mode 2.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (TIM_TypeDef * TIMx)**
Function description	Indicate whether external clock mode 2 is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name	**__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)**
Function description	Set the clock source of the counter clock.
Parameters	• **TIMx:** Timer instance • **ClockSource:** This parameter can be one of the following values: – LL_TIM_CLOCKSOURCE_INTERNAL – LL_TIM_CLOCKSOURCE_EXT_MODE1 – LL_TIM_CLOCKSOURCE_EXT_MODE2
Return values	• **None:**
Notes	• when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function. • Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1. • Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.
Reference Manual to LL API cross reference:	• SMCR SMS LL_TIM_SetClockSource • SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name	**__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)**
Function description	Set the encoder interface mode.

Parameters
- **TIMx:** Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_TIM_ENCODERMODE_X2_TI1
  - LL_TIM_ENCODERMODE_X2_TI2
  - LL_TIM_ENCODERMODE_X4_TI12

Return values
- **None:**

Notes
- Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

Reference Manual to LL API cross reference:
- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name
__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)

Function description
Set the trigger output (TRGO) used for timer synchronization .

Parameters
- **TIMx:** Timer instance
- **TimerSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO_RESET
  - LL_TIM_TRGO_ENABLE
  - LL_TIM_TRGO_UPDATE
  - LL_TIM_TRGO_CC1IF
  - LL_TIM_TRGO_OC1REF
  - LL_TIM_TRGO_OC2REF
  - LL_TIM_TRGO_OC3REF
  - LL_TIM_TRGO_OC4REF

Return values
- **None:**

Notes
- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.

Reference Manual to LL API cross reference:
- CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetSlaveMode

Function name
__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)

Function description
Set the synchronization mode of a slave timer.

Parameters
- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  - LL_TIM_SLAVEMODE_DISABLED
  - LL_TIM_SLAVEMODE_RESET

- LL_TIM_SLAVEMODE_GATED
- LL_TIM_SLAVEMODE_TRIGGER

Return values
- **None:**

Notes
- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:
- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name
__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)

Function description
Set the selects the trigger input to be used to synchronize the counter.

Parameters
- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  - LL_TIM_TS_ITR0
  - LL_TIM_TS_ITR1
  - LL_TIM_TS_ITR2
  - LL_TIM_TS_ITR3
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF

Return values
- **None:**

Notes
- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:
- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name
__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description
Enable the Master/Slave mode.

Parameters
- **TIMx:** Timer instance

Return values
- **None:**

Notes
- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross
- SMCR MSM LL_TIM_EnableMasterSlaveMode

reference:

LL_TIM_DisableMasterSlaveMode

Function name	**__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)**
Function description	Disable the Master/Slave mode.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (TIM_TypeDef * TIMx)**
Function description	Indicates whether the Master/Slave mode is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.
Reference Manual to LL API cross reference:	• SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name	**__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)**
Function description	Configure the external trigger (ETR) input.
Parameters	• **TIMx:** Timer instance • **ETRPolarity:** This parameter can be one of the following values: – LL_TIM_ETR_POLARITY_NONINVERTED – LL_TIM_ETR_POLARITY_INVERTED • **ETRPrescaler:** This parameter can be one of the following values: – LL_TIM_ETR_PRESCALER_DIV1 – LL_TIM_ETR_PRESCALER_DIV2 – LL_TIM_ETR_PRESCALER_DIV4 – LL_TIM_ETR_PRESCALER_DIV8 • **ETRFilter:** This parameter can be one of the following

values:
- LL_TIM_ETR_FILTER_FDIV1
- LL_TIM_ETR_FILTER_FDIV1_N2
- LL_TIM_ETR_FILTER_FDIV1_N4
- LL_TIM_ETR_FILTER_FDIV1_N8
- LL_TIM_ETR_FILTER_FDIV2_N6
- LL_TIM_ETR_FILTER_FDIV2_N8
- LL_TIM_ETR_FILTER_FDIV4_N6
- LL_TIM_ETR_FILTER_FDIV4_N8
- LL_TIM_ETR_FILTER_FDIV8_N6
- LL_TIM_ETR_FILTER_FDIV8_N8
- LL_TIM_ETR_FILTER_FDIV16_N5
- LL_TIM_ETR_FILTER_FDIV16_N6
- LL_TIM_ETR_FILTER_FDIV16_N8
- LL_TIM_ETR_FILTER_FDIV32_N5
- LL_TIM_ETR_FILTER_FDIV32_N6
- LL_TIM_ETR_FILTER_FDIV32_N8

Return values
- **None:**

Notes
- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:
- SMCR ETP LL_TIM_ConfigETR
- SMCR ETPS LL_TIM_ConfigETR
- SMCR ETF LL_TIM_ConfigETR

LL_TIM_EnableBRK

Function name **__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)**

Function description Enable the break function.

Parameters
- **TIMx:** Timer instance

Return values
- **None:**

Notes
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name **__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)**

Function description Disable the break function.

Parameters
- **TIMx:** Timer instance

Return values
- **None:**

Notes
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR BKE LL_TIM_DisableBRK

LL_TIM_ConfigBRK

Function name: **__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity)**

Function description: Configure the break input.

Parameters:
- **TIMx:** Timer instance
- **BreakPolarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK_POLARITY_LOW
  - LL_TIM_BREAK_POLARITY_HIGH

Return values:
- **None:**

Notes:
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR BKP LL_TIM_ConfigBRK

LL_TIM_SetOffStates

Function name: **__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)**

Function description: Select the outputs off state (enabled v.s.

Parameters:
- **TIMx:** Timer instance
- **OffStateIdle:** This parameter can be one of the following values:
  - LL_TIM_OSSI_DISABLE
  - LL_TIM_OSSI_ENABLE
- **OffStateRun:** This parameter can be one of the following values:
  - LL_TIM_OSSR_DISABLE
  - LL_TIM_OSSR_ENABLE

Return values:
- **None:**

Notes:
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR OSSI LL_TIM_SetOffStates
- BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name: **__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)**

Function description: Enable automatic output (MOE can be set by software or automatically when a break input is active).

Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name	**__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)**
Function description	Disable automatic output (MOE can be set only by software).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (TIM_TypeDef * TIMx)**
Function description	Indicate whether automatic output is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
Reference Manual to LL API cross reference:	• BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name	**__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)**
Function description	Enable the outputs (set the MOE bit in TIMx_BDTR register).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Notes	• The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event • Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to

check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name **__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)**

Function description Disable the outputs (reset the MOE bit in TIMx_BDTR register).

Parameters
- **TIMx:** Timer instance

Return values
- **None:**

Notes
- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name **__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (TIM_TypeDef * TIMx)**

Function description Indicates whether outputs are enabled.

Parameters
- **TIMx:** Timer instance

Return values
- **State:** of bit (1 or 0).

Notes
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:
- BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_ConfigDMABurst

Function name **__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength)**

Function description Configures the timer DMA burst feature.

Parameters
- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER

  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
- **DMABurstLength:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_LENGTH_1TRANSFER
  - LL_TIM_DMABURST_LENGTH_2TRANSFERS
  - LL_TIM_DMABURST_LENGTH_3TRANSFERS
  - LL_TIM_DMABURST_LENGTH_4TRANSFERS
  - LL_TIM_DMABURST_LENGTH_5TRANSFERS
  - LL_TIM_DMABURST_LENGTH_6TRANSFERS
  - LL_TIM_DMABURST_LENGTH_7TRANSFERS
  - LL_TIM_DMABURST_LENGTH_8TRANSFERS
  - LL_TIM_DMABURST_LENGTH_9TRANSFERS
  - LL_TIM_DMABURST_LENGTH_10TRANSFERS
  - LL_TIM_DMABURST_LENGTH_11TRANSFERS
  - LL_TIM_DMABURST_LENGTH_12TRANSFERS
  - LL_TIM_DMABURST_LENGTH_13TRANSFERS
  - LL_TIM_DMABURST_LENGTH_14TRANSFERS
  - LL_TIM_DMABURST_LENGTH_15TRANSFERS
  - LL_TIM_DMABURST_LENGTH_16TRANSFERS
  - LL_TIM_DMABURST_LENGTH_17TRANSFERS
  - LL_TIM_DMABURST_LENGTH_18TRANSFERS

Return values
- **None:**

Notes
- Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.

Reference Manual to LL API cross reference:
- DCR DBL LL_TIM_ConfigDMABurst
- DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_ClearFlag_UPDATE

Function name
__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)

Function description
Clear the update interrupt flag (UIF).

Parameters
- **TIMx:** Timer instance

Return values
- **None:**

Reference Manual to LL API cross reference:	• SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 1 interrupt flag (CC1F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 2 interrupt flag (CC2F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 3 interrupt flag (CC3F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 4 interrupt flag (CC4F).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_COM

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)**
Function description	Clear the commutation interrupt flag (COMIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (TIM_TypeDef * TIMx)**
Function description	Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)**
Function description	Clear the trigger interrupt flag (TIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)**
Function description	Clear the break interrupt flag (BIF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (TIM_TypeDef * TIMx)**
Function description	Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_CC1OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**

Reference Manual to LL API cross reference:	• SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).
Parameters	• **TIMx:** Timer instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name	**__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)**
Function description	Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (TIM_TypeDef * TIMx)**
Function description	Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_EnableIT_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Enable update interrupt (UIE).

Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Disable update interrupt (UIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicates whether the update interrupt (UIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 1 interrupt (CC1IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 1 interrupt (CC1IE).
Parameters	• **TIMx:** Timer instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 2 interrupt (CC2IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 2 interrupt (CC2IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.
Parameters	• **TIMx:** Timer instance

Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 3 interrupt (CC3IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 3 interrupt (CC3IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 4 interrupt (CC4IE).
Parameters	• **TIMx:** Timer instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 4 interrupt (CC4IE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)**
Function description	Enable commutation interrupt (COMIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)**
Function description	Disable commutation interrupt (COMIE).
Parameters	• **TIMx:** Timer instance

Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (TIM_TypeDef * TIMx)**
Function description	Indicates whether the commutation interrupt (COMIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Enable trigger interrupt (TIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Disable trigger interrupt (TIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicates whether the trigger interrupt (TIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name	**__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)**
Function description	Enable break interrupt (BIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name	**__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)**
Function description	Disable break interrupt (BIE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (TIM_TypeDef * TIMx)**
Function description	Indicates whether the break interrupt (BIE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableDMAReq_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Enable update DMA request (UDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross	• DIER UDE LL_TIM_EnableDMAReq_UPDATE

reference:

LL_TIM_DisableDMAReq_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Disable update DMA request (UDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (TIM_TypeDef * TIMx)**
Function description	Indicates whether the update DMA request (UDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 1 DMA request (CC1DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 1 DMA request (CC1DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 2 DMA request (CC2DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 2 DMA request (CC2DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 3 DMA request (CC3DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 3 DMA request (CC3DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Enable capture/compare 4 DMA request (CC4DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Disable capture/compare 4 DMA request (CC4DE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (TIM_TypeDef * TIMx)**
Function description	Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)**
Function description	Enable commutation DMA request (COMDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)**
Function description	Disable commutation DMA request (COMDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (TIM_TypeDef * TIMx)**
Function description	Indicates whether the commutation DMA request (COMDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name	**__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Enable trigger interrupt (TDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name	**__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Disable trigger interrupt (TDE).
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name	**__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (TIM_TypeDef * TIMx)**
Function description	Indicates whether the trigger interrupt (TDE) is enabled.
Parameters	• **TIMx:** Timer instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)**
Function description	Generate an update event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 1 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 2 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 3 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)**
Function description	Generate Capture/Compare 4 event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)**
Function description	Generate commutation event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)**
Function description	Generate trigger event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name	**__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)**
Function description	Generate break event.
Parameters	• **TIMx:** Timer instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_DeInit

Function name **ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)**

Function description Set TIMx registers to their reset values.

Parameters
- **TIMx:** Timer instance

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx registers are de-initialized
  – ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name **void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)**

Function description Set the fields of the time base unit configuration data structure to their default values.

Parameters
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)

Return values
- **None:**

LL_TIM_Init

Function name **ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, LL_TIM_InitTypeDef * TIM_InitStruct)**

Function description Configure the TIMx time base unit.

Parameters
- **TIMx:** Timer Instance
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx registers are de-initialized
  – ERROR: not applicable

LL_TIM_OC_StructInit

Function name **void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)**

Function description Set the fields of the TIMx output channel configuration data structure to their default values.

Parameters
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)

Return values
- **None:**

LL_TIM_OC_Init

Function name **ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)**

Function description Configure the TIMx output channel.

Parameters
- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:

  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name **void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)**

Function description Set the fields of the TIMx input channel configuration data structure to their default values.

Parameters
- **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)

Return values
- **None:**

LL_TIM_IC_Init

Function name **ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)**

Function description Configure the TIMx input channel.

Parameters
- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name **void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)**

Function description Fills each TIM_EncoderInitStruct field with its default value.

Parameters
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)

Return values
- **None:**

LL_TIM_ENCODER_Init

Function name **ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)**

Function description Configure the encoder interface of the timer instance.

Parameters
- **TIMx:** Timer Instance
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx registers are de-initialized
  – ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name **void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)**

Function description Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.

Parameters
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values
- **None:**

LL_TIM_HALLSENSOR_Init

Function name **ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)**

Function description Configure the Hall sensor interface of the timer instance.

Parameters
- **TIMx:** Timer Instance
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx registers are de-initialized
  – ERROR: not applicable

Notes
- TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel
- TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.
- Channel 1 is configured as input, IC1 is mapped on TRC.
- Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.
- Channel 2 is configured in output PWM 2 mode.
- Compare value stored in TIMx_CCR2 corresponds to the commutation delay.

- OC2REF is selected as trigger output on TRGO.

LL_TIM_BDTR_StructInit

Function name	**void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)**
Function description	Set the fields of the Break and Dead Time configuration data structure to their default values.
Parameters	• **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)
Return values	• **None:**

LL_TIM_BDTR_Init

Function name	**ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)**
Function description	Configure the Break and Dead Time feature of the timer instance.
Parameters	• **TIMx:** Timer Instance • **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure(Break and Dead Time configuration data structure)
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: Break and Dead Time is initialized – ERROR: not applicable
Notes	• As the bits AOE, BKP, BKE, OSSR, OSSI and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register. • Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

62.3 TIM Firmware driver defines

62.3.1 TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI	ICx is mapped on TIx
LL_TIM_ACTIVEINPUT_INDIRECTTI	ICx is mapped on TIy
LL_TIM_ACTIVEINPUT_TRC	ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE	MOE can be set only by software
LL_TIM_AUTOMATICOUTPUT_ENABLE	MOE can be set by software or automatically at the next update event

Break Enable

LL_TIM_BREAK_DISABLE	Break function disabled

LL_TIM_BREAK_ENABLE	Break function enabled

break polarity

LL_TIM_BREAK_POLARITY_LOW	Break input BRK is active low
LL_TIM_BREAK_POLARITY_HIGH	Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC	CCx DMA request sent when CCx event occurs
LL_TIM_CCDMAREQUEST_UPDATE	CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY	Capture/compare control bits are updated by setting the COMG bit only
LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI	Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_TIM_CHANNEL_CH1	Timer input/output channel 1
LL_TIM_CHANNEL_CH1N	Timer complementary output channel 1
LL_TIM_CHANNEL_CH2	Timer input/output channel 2
LL_TIM_CHANNEL_CH2N	Timer complementary output channel 2
LL_TIM_CHANNEL_CH3	Timer input/output channel 3
LL_TIM_CHANNEL_CH3N	Timer complementary output channel 3
LL_TIM_CHANNEL_CH4	Timer input/output channel 4

Clock Division

LL_TIM_CLOCKDIVISION_DIV1	tDTS=tCK_INT
LL_TIM_CLOCKDIVISION_DIV2	tDTS=2*tCK_INT
LL_TIM_CLOCKDIVISION_DIV4	tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL	The timer is clocked by the internal clock provided from the RCC
LL_TIM_CLOCKSOURCE_EXT_MODE1	Counter counts at each rising or falling edge on a selected inpu t
LL_TIM_CLOCKSOURCE_EXT_MODE2	Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP	Timer counter counts up
LL_TIM_COUNTERDIRECTION_DOWN	Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP	Counter used as upcounter
LL_TIM_COUNTERMODE_DOWN	Counter used as downcounter
LL_TIM_COUNTERMODE_CENTER_UP	The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.
LL_TIM_COUNTERMODE_CENTER_DOWN	The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up
LL_TIM_COUNTERMODE_CENTER_UP_DOWN	The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1	TIMx_CR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CR2	TIMx_CR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_SMCR	TIMx_SMCR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_DIER	TIMx_DIER register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_SR	TIMx_SR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_EGR	TIMx_EGR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR1	TIMx_CCMR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCMR2	TIMx_CCMR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCER	TIMx_CCER register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CNT	TIMx_CNT register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_PSC	TIMx_PSC register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_ARR	TIMx_ARR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_RCR	TIMx_RCR register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR1	TIMx_CCR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR2	TIMx_CCR2 register is the DMA base

	address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR3	TIMx_CCR3 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR4	TIMx_CCR4 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_BDTR	TIMx_BDTR register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER	Transfer is done to 1 register starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_2TRANSFERS	Transfer is done to 2 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_3TRANSFERS	Transfer is done to 3 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_4TRANSFERS	Transfer is done to 4 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_5TRANSFERS	Transfer is done to 5 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_6TRANSFERS	Transfer is done to 6 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_7TRANSFERS	Transfer is done to 7 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_8TRANSFERS	Transfer is done to 1 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_9TRANSFERS	Transfer is done to 9 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_10TRANSFERS	Transfer is done to 10 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_11TRANSFERS	Transfer is done to 11 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_12TRANSFERS	Transfer is done to 12 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_13TRANSFERS	Transfer is done to 13 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_14TRANSFERS	Transfer is done to 14 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_15TRANSFERS	Transfer is done to 15 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_16TRANSFERS	Transfer is done to 16 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_17TRANSFERS	Transfer is done to 17 registers starting from the DMA burst base address
LL_TIM_DMABURST_LENGTH_18TRANSFERS	Transfer is done to 18 registers starting from the DMA burst base address

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1	Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level
LL_TIM_ENCODERMODE_X2_TI2	Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level
LL_TIM_ENCODERMODE_X4_TI12	Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input l

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1	No filter, sampling is done at fDTS
LL_TIM_ETR_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_ETR_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_ETR_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_ETR_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_ETR_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_ETR_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_ETR_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_ETR_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=8
LL_TIM_ETR_FILTER_FDIV8_N8	fSAMPLING=fDTS/16, N=5
LL_TIM_ETR_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=6
LL_TIM_ETR_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=8
LL_TIM_ETR_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=5
LL_TIM_ETR_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_ETR_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_ETR_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED	ETR is non-inverted, active at high level or rising edge
LL_TIM_ETR_POLARITY_INVERTED	ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1	ETR prescaler OFF
LL_TIM_ETR_PRESCALER_DIV2	ETR frequency is divided by 2
LL_TIM_ETR_PRESCALER_DIV4	ETR frequency is divided by 4
LL_TIM_ETR_PRESCALER_DIV8	ETR frequency is divided by 8

Get Flags Defines

LL_TIM_SR_UIF	Update interrupt flag
LL_TIM_SR_CC1IF	Capture/compare 1 interrupt flag

LL_TIM_SR_CC2IF	Capture/compare 2 interrupt flag
LL_TIM_SR_CC3IF	Capture/compare 3 interrupt flag
LL_TIM_SR_CC4IF	Capture/compare 4 interrupt flag
LL_TIM_SR_COMIF	COM interrupt flag
LL_TIM_SR_TIF	Trigger interrupt flag
LL_TIM_SR_BIF	Break interrupt flag
LL_TIM_SR_CC1OF	Capture/Compare 1 overcapture flag
LL_TIM_SR_CC2OF	Capture/Compare 2 overcapture flag
LL_TIM_SR_CC3OF	Capture/Compare 3 overcapture flag
LL_TIM_SR_CC4OF	Capture/Compare 4 overcapture flag

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1	No prescaler, capture is done each time an edge is detected on the capture input
LL_TIM_ICPSC_DIV2	Capture is done once every 2 events
LL_TIM_ICPSC_DIV4	Capture is done once every 4 events
LL_TIM_ICPSC_DIV8	Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1	No filter, sampling is done at fDTS
LL_TIM_IC_FILTER_FDIV1_N2	fSAMPLING=fCK_INT, N=2
LL_TIM_IC_FILTER_FDIV1_N4	fSAMPLING=fCK_INT, N=4
LL_TIM_IC_FILTER_FDIV1_N8	fSAMPLING=fCK_INT, N=8
LL_TIM_IC_FILTER_FDIV2_N6	fSAMPLING=fDTS/2, N=6
LL_TIM_IC_FILTER_FDIV2_N8	fSAMPLING=fDTS/2, N=8
LL_TIM_IC_FILTER_FDIV4_N6	fSAMPLING=fDTS/4, N=6
LL_TIM_IC_FILTER_FDIV4_N8	fSAMPLING=fDTS/4, N=8
LL_TIM_IC_FILTER_FDIV8_N6	fSAMPLING=fDTS/8, N=6
LL_TIM_IC_FILTER_FDIV8_N8	fSAMPLING=fDTS/8, N=8
LL_TIM_IC_FILTER_FDIV16_N5	fSAMPLING=fDTS/16, N=5
LL_TIM_IC_FILTER_FDIV16_N6	fSAMPLING=fDTS/16, N=6
LL_TIM_IC_FILTER_FDIV16_N8	fSAMPLING=fDTS/16, N=8
LL_TIM_IC_FILTER_FDIV32_N5	fSAMPLING=fDTS/32, N=5
LL_TIM_IC_FILTER_FDIV32_N6	fSAMPLING=fDTS/32, N=6
LL_TIM_IC_FILTER_FDIV32_N8	fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING	The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted
LL_TIM_IC_POLARITY_FALLING	The circuit is sensitive to TIxFP1 falling edge, TIxFP1

is inverted

IT Defines

LL_TIM_DIER_UIE	Update interrupt enable
LL_TIM_DIER_CC1IE	Capture/compare 1 interrupt enable
LL_TIM_DIER_CC2IE	Capture/compare 2 interrupt enable
LL_TIM_DIER_CC3IE	Capture/compare 3 interrupt enable
LL_TIM_DIER_CC4IE	Capture/compare 4 interrupt enable
LL_TIM_DIER_COMIE	COM interrupt enable
LL_TIM_DIER_TIE	Trigger interrupt enable
LL_TIM_DIER_BIE	Break interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF	LOCK OFF - No bit is write protected
LL_TIM_LOCKLEVEL_1	LOCK Level 1
LL_TIM_LOCKLEVEL_2	LOCK Level 2
LL_TIM_LOCKLEVEL_3	LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW	OCx=0 (after a dead-time if OC is implemented) when MOE=0
LL_TIM_OCIDLESTATE_HIGH	OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN	The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level
LL_TIM_OCMODE_ACTIVE	OCyREF is forced high on compare match
LL_TIM_OCMODE_INACTIVE	OCyREF is forced low on compare match
LL_TIM_OCMODE_TOGGLE	OCyREF toggles on compare match
LL_TIM_OCMODE_FORCED_INACTIVE	OCyREF is forced low
LL_TIM_OCMODE_FORCED_ACTIVE	OCyREF is forced high
LL_TIM_OCMODE_PWM1	In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.
LL_TIM_OCMODE_PWM2	In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH	OCxactive high

LL_TIM_OCPOLARITY_LOW	OCxactive low

Output Configuration State

LL_TIM_OCSTATE_DISABLE	OCx is not active
LL_TIM_OCSTATE_ENABLE	OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE	Counter is not stopped at update event
LL_TIM_ONEPULSEMODE_REPETITIVE	Counter stops counting at the next update event

OSSI

LL_TIM_OSSI_DISABLE	When inactive, OCx/OCxN outputs are disabled
LL_TIM_OSSI_ENABLE	When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE	When inactive, OCx/OCxN outputs are disabled
LL_TIM_OSSR_ENABLE	When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Slave Mode

LL_TIM_SLAVEMODE_DISABLED	Slave mode disabled
LL_TIM_SLAVEMODE_RESET	Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter
LL_TIM_SLAVEMODE_GATED	Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high
LL_TIM_SLAVEMODE_TRIGGER	Trigger Mode - The counter starts at a rising edge of the trigger TRGI

Trigger Output

LL_TIM_TRGO_RESET	UG bit from the TIMx_EGR register is used as trigger output
LL_TIM_TRGO_ENABLE	Counter Enable signal (CNT_EN) is used as trigger output
LL_TIM_TRGO_UPDATE	Update event is used as trigger output
LL_TIM_TRGO_CC1IF	CC1 capture or a compare match is used as trigger output
LL_TIM_TRGO_OC1REF	OC1REF signal is used as trigger output
LL_TIM_TRGO_OC2REF	OC2REF signal is used as trigger output
LL_TIM_TRGO_OC3REF	OC3REF signal is used as trigger output
LL_TIM_TRGO_OC4REF	OC4REF signal is used as trigger output

Trigger Selection

LL_TIM_TS_ITR0	Internal Trigger 0 (ITR0) is used as trigger input
LL_TIM_TS_ITR1	Internal Trigger 1 (ITR1) is used as trigger input
LL_TIM_TS_ITR2	Internal Trigger 2 (ITR2) is used as trigger input
LL_TIM_TS_ITR3	Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_TI1F_ED	TI1 Edge Detector (TI1F_ED) is used as trigger input
LL_TIM_TS_TI1FP1	Filtered Timer Input 1 (TI1FP1) is used as trigger input
LL_TIM_TS_TI2FP2	Filtered Timer Input 2 (TI12P2) is used as trigger input
LL_TIM_TS_ETRF	Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR	Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request
LL_TIM_UPDATESOURCE_COUNTER	Only counter overflow/underflow generates an update request

Exported_Macros

__LL_TIM_CALC_DEADTIME

Description:

- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:

- DTG[0:7]

Notes:

- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value

to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

Common Write and read registers Macros

LL_TIM_WriteReg

Description:

- Write a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_TIM_ReadReg

Description:

- Read a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:

- Register: value

63 LL USART Generic Driver

63.1 USART Firmware driver registers structures

63.1.1 LL_USART_InitTypeDef

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::BaudRate***
  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.

63.1.2 LL_USART_ClockInitTypeDef

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t LastBitClockPulse***

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***
  Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_LL_EC_CLOCK***.USART HW configuration can be modified

afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.

- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_LL_EC_POLARITY***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_LL_EC_PHASE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***
  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_LL_EC_LASTCLKPULSE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

63.2 USART Firmware driver API description

63.2.1 Detailed description of functions

LL_USART_Enable

Function name	**__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)**
Function description	USART Enable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 UE LL_USART_Enable

LL_USART_Disable

Function name	**__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)**
Function description	USART Disable (all USART prescalers and outputs are disabled)
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_SR are set to their default values.
Reference Manual to LL API cross reference:	• CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabled (USART_TypeDef * USARTx)**
Function description	Indicate if USART is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 UE LL_USART_IsEnabled

LL_USART_EnableDirectionRx

Function name	**__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)**
Function description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name	**__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)**
Function description	Receiver Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name	**__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)**
Function description	Transmitter Enable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name	**__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)**
Function description	Transmitter Disable.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name	**__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)**
Function description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	• **USARTx:** USART Instance • **TransferDirection:** This parameter can be one of the following values: – LL_USART_DIRECTION_NONE – LL_USART_DIRECTION_RX – LL_USART_DIRECTION_TX – LL_USART_DIRECTION_TX_RX
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_SetTransferDirection • CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name	**__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (USART_TypeDef * USARTx)**
Function description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DIRECTION_NONE – LL_USART_DIRECTION_RX – LL_USART_DIRECTION_TX – LL_USART_DIRECTION_TX_RX
Reference Manual to LL API cross reference:	• CR1 RE LL_USART_GetTransferDirection • CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name	**__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)**

Function description	Configure Parity (enabled/disabled and parity mode if enabled).
Parameters	• **USARTx:** USART Instance • **Parity:** This parameter can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD
Return values	• **None:**
Notes	• This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_SetParity • CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name	**__STATIC_INLINE uint32_t LL_USART_GetParity (USART_TypeDef * USARTx)**
Function description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_PARITY_NONE – LL_USART_PARITY_EVEN – LL_USART_PARITY_ODD
Reference Manual to LL API cross reference:	• CR1 PS LL_USART_GetParity • CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name	**__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)**
Function description	Set Receiver Wake Up method from Mute mode.
Parameters	• **USARTx:** USART Instance • **Method:** This parameter can be one of the following values: – LL_USART_WAKEUP_IDLELINE – LL_USART_WAKEUP_ADDRESSMARK
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name	**__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (USART_TypeDef * USARTx)**
Function description	Return Receiver Wake Up method from Mute mode.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_WAKEUP_IDLELINE – LL_USART_WAKEUP_ADDRESSMARK
Reference Manual to LL API cross reference:	• CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name	**__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)**
Function description	Set Word length (i.e.
Parameters	• **USARTx:** USART Instance • **DataWidth:** This parameter can be one of the following values: – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 M LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name	**__STATIC_INLINE uint32_t LL_USART_GetDataWidth (USART_TypeDef * USARTx)**
Function description	Return Word length (i.e.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_DATAWIDTH_8B – LL_USART_DATAWIDTH_9B
Reference Manual to LL API cross reference:	• CR1 M LL_USART_GetDataWidth

LL_USART_SetLastClkPulseOutput

Function name	**__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)**
Function description	Configure if Clock pulse of the last data bit is output to the SCLK pin or not.

Parameters	• **USARTx:** USART Instance • **LastBitClockPulse:** This parameter can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name	**__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (USART_TypeDef * USARTx)**
Function description	Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_LASTCLKPULSE_NO_OUTPUT – LL_USART_LASTCLKPULSE_OUTPUT
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name	**__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)**
Function description	Select the phase of the clock output on the SCLK pin in synchronous mode.
Parameters	• **USARTx:** USART Instance • **ClockPhase:** This parameter can be one of the following values: – LL_USART_PHASE_1EDGE – LL_USART_PHASE_2EDGE
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to	• CR2 CPHA LL_USART_SetClockPhase

LL API cross reference:

LL_USART_GetClockPhase

Function name
__STATIC_INLINE uint32_t LL_USART_GetClockPhase (USART_TypeDef * USARTx)

Function description
Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters
- **USARTx:** USART Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:
- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name
__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description
Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters
- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values
- **None:**

Notes
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:
- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name
__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (USART_TypeDef * USARTx)

Function description
Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters
- **USARTx:** USART Instance

Return values
- **Returned:** value can be one of the following values:

  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:
- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name
__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description
Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters
- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values
- **None:**

Notes
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:
- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_EnableSCLKOutput

Function name
__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)

Function description
Enable Clock output on SCLK pin.

Parameters
- **USARTx:** USART Instance

Return values
- **None:**

Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name	**__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)**
Function description	Disable Clock output on SCLK pin.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (USART_TypeDef * USARTx)**
Function description	Indicate if Clock output on SCLK pin is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name	**__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)**
Function description	Set the length of the stop bits.
Parameters	• **USARTx:** USART Instance • **StopBits:** This parameter can be one of the following values: – LL_USART_STOPBITS_0_5 – LL_USART_STOPBITS_1 – LL_USART_STOPBITS_1_5 – LL_USART_STOPBITS_2

Return values
- **None:**

Reference Manual to LL API cross reference:
- CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name
__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (USART_TypeDef * USARTx)

Function description
Retrieve the length of the stop bits.

Parameters
- **USARTx:** USART Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Reference Manual to LL API cross reference:
- CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name
__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description
Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters
- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values
- **None:**

Notes
- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function

Reference Manual to LL API cross
- CR1 PS LL_USART_ConfigCharacter

reference:
- CR1 PCE LL_USART_ConfigCharacter
- CR1 M LL_USART_ConfigCharacter
- CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetNodeAddress

Function name	**__STATIC_INLINE void LL_USART_SetNodeAddress (USART_TypeDef * USARTx, uint32_t NodeAddress)**
Function description	Set Address of the USART node.
Parameters	• **USARTx:** USART Instance • **NodeAddress:** 4 bit Address of the USART node.
Return values	• **None:**
Notes	• This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
Reference Manual to LL API cross reference:	• CR2 ADD LL_USART_SetNodeAddress

LL_USART_GetNodeAddress

Function name	**__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (USART_TypeDef * USARTx)**
Function description	Return 4 bit Address of the USART node as set in ADD field of CR2.
Parameters	• **USARTx:** USART Instance
Return values	• **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)
Notes	• only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant)
Reference Manual to LL API cross reference:	• CR2 ADD LL_USART_GetNodeAddress

LL_USART_EnableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Enable RTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross	• CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

reference:

LL_USART_DisableRTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Disable RTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Enable CTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name	**__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)**
Function description	Disable CTS HW Flow Control.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name
__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)

Function description
Configure HW Flow Control mode (both CTS and RTS)

Parameters
- **USARTx:** USART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Return values
- **None:**

Notes
- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:
- CR3 RTSE LL_USART_SetHWFlowCtrl
- CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name
__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (USART_TypeDef * USARTx)

Function description
Return HW Flow Control configuration (both CTS and RTS)

Parameters
- **USARTx:** USART Instance

Return values
- **Returned:** value can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Notes
- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:
- CR3 RTSE LL_USART_GetHWFlowCtrl
- CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_SetBaudRate

Function name
__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t BaudRate)

Function description
Configure USART BRR register for achieving expected Baud Rate value.

Parameters
- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **BaudRate:** Baud Rate

Return values	• **None:**
Notes	• Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values • Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
Reference Manual to LL API cross reference:	• BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name	**__STATIC_INLINE uint32_t LL_USART_GetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk)**
Function description	Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.
Parameters	• **USARTx:** USART Instance • **PeriphClk:** Peripheral Clock
Return values	• **Baud:** Rate
Notes	• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
Reference Manual to LL API cross reference:	• BRR BRR LL_USART_GetBaudRate

LL_USART_EnableIrda

Function name	**__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)**
Function description	Enable IrDA mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name	**__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)**
Function description	Disable IrDA mode.
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (USART_TypeDef * USARTx)**
Function description	Indicate if IrDA mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name	**__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)**
Function description	Configure IrDA Power Mode (Normal or Low Power)
Parameters	• **USARTx:** USART Instance • **PowerMode:** This parameter can be one of the following values: – LL_USART_IRDA_POWER_NORMAL – LL_USART_IRDA_POWER_LOW
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name	**__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (USART_TypeDef * USARTx)**
Function description	Retrieve IrDA Power Mode configuration (Normal or Low Power)
Parameters	• **USARTx:** USART Instance

Return values	• **Returned:** value can be one of the following values: – LL_USART_IRDA_POWER_NORMAL – LL_USART_PHASE_2EDGE
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (USART_TypeDef * USARTx)**
Function description	Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	• **USARTx:** USART Instance
Return values	• **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name	**__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)**

Function description	Enable Smartcard NACK transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name	**__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Disable Smartcard NACK transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (USART_TypeDef * USARTx)**
Function description	Indicate if Smartcard NACK transmission is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name	**__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)**
Function description	Enable Smartcard mode.
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name	**__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)**
Function description	Disable Smartcard mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (USART_TypeDef * USARTx)**
Function description	Indicate if Smartcard mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardPrescaler

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)**
Function description	Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	• **USARTx:** USART Instance • **PrescalerValue:** Value between Min_Data=0 and Max_Data=31
Return values	• **None:**

Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (USART_TypeDef * USARTx)**
Function description	Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name	**__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)**
Function description	Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)
Parameters	• **USARTx:** USART Instance • **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name	**__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (USART_TypeDef * USARTx)**
Function description	Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters	• **USARTx:** USART Instance
Return values	• **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)
Notes	• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name	**__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)**
Function description	Enable Single Wire Half-Duplex mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name	**__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)**
Function description	Disable Single Wire Half-Duplex mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (USART_TypeDef * USARTx)**
Function description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Notes	• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_SetLINBrkDetectionLen

Function name	**__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)**
Function description	Set LIN Break Detection Length.
Parameters	• **USARTx:** USART Instance • **LINBDLength:** This parameter can be one of the following values: – LL_USART_LINBREAK_DETECT_10B – LL_USART_LINBREAK_DETECT_11B
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name	**__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (USART_TypeDef * USARTx)**
Function description	Return LIN Break Detection Length.
Parameters	• **USARTx:** USART Instance
Return values	• **Returned:** value can be one of the following values: – LL_USART_LINBREAK_DETECT_10B – LL_USART_LINBREAK_DETECT_11B
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name	**__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)**
Function description	Enable LIN mode.

Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name	**__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)**
Function description	Disable LIN mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (USART_TypeDef * USARTx)**
Function description	Indicate if LIN mode is enabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_ConfigAsyncMode

Function name	**__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)
Parameters	• **USARTx:** USART Instance
Return values	• **None:**

Notes
- In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:
- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name **__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)**

Function description Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters
- **USARTx:** USART Instance

Return values
- **None:**

Notes
- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:
- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode

- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name	**__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in LIN Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode. • Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance. • Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function • Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions
Reference Manual to LL API cross reference:	• CR2 CLKEN LL_USART_ConfigLINMode • CR2 STOP LL_USART_ConfigLINMode • CR2 LINEN LL_USART_ConfigLINMode • CR3 IREN LL_USART_ConfigLINMode • CR3 SCEN LL_USART_ConfigLINMode • CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name	**__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)**
Function description	Perform basic configuration of USART for enabling use in Half Duplex Mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3

register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:
- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name: **__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)**

Function description: Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters:
- **USARTx:** USART Instance

Return values:
- **None:**

Notes:
- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name **__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)**

Function description Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name **__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)**

Function description Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values
- **None:**

Notes
- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:
- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name **__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (USART_TypeDef * USARTx)**

Function description Check if the USART Parity Error Flag is set or not.

Parameters
- **USARTx:** USART Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- SR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name **__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (USART_TypeDef * USARTx)**

Function description Check if the USART Framing Error Flag is set or not.

Parameters
- **USARTx:** USART Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- SR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name **__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE**

(USART_TypeDef * USARTx)

Function description	Check if the USART Noise error detected Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR NF LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (USART_TypeDef * USARTx)**
Function description	Check if the USART OverRun Error Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (USART_TypeDef * USARTx)**
Function description	Check if the USART IDLE line detected Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE (USART_TypeDef * USARTx)**
Function description	Check if the USART Read Data Register Not Empty Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR RXNE LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TC

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC**

(USART_TypeDef * USARTx)

Function description	Check if the USART Transmission Complete Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmit Data Register Empty Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• SR TXE LL_USART_IsActiveFlag_TXE

LL_USART_IsActiveFlag_LBD

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (USART_TypeDef * USARTx)**
Function description	Check if the USART LIN Break Detection Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• SR LBD LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (USART_TypeDef * USARTx)**
Function description	Check if the USART CTS Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:	• SR CTS LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_SBK

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (USART_TypeDef * USARTx)**
Function description	Check if the USART Send Break Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 SBK LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name	**__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (USART_TypeDef * USARTx)**
Function description	Check if the USART Receive Wake Up from mute mode Flag is set or not.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RWU LL_USART_IsActiveFlag_RWU

LL_USART_ClearFlag_PE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)**
Function description	Clear Parity Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the USARTx_SR register followed by a read access to the USARTx_DR register. • Please also consider that when clearing this flag, other flags as NE, FE, ORE, IDLE would also be cleared.
Reference Manual to LL API cross reference:	• SR PE LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)**

Function description	Clear Framing Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the USARTx_SR register followed by a read access to the USARTx_DR register. • Please also consider that when clearing this flag, other flags as PE, NE, ORE, IDLE would also be cleared.
Reference Manual to LL API cross reference:	• SR FE LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)**
Function description	Clear Noise detected Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the USARTx_SR register followed by a read access to the USARTx_DR register. • Please also consider that when clearing this flag, other flags as PE, FE, ORE, IDLE would also be cleared.
Reference Manual to LL API cross reference:	• SR NF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)**
Function description	Clear OverRun Error Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the USARTx_SR register followed by a read access to the USARTx_DR register. • Please also consider that when clearing this flag, other flags as PE, NE, FE, IDLE would also be cleared.
Reference Manual to LL API cross reference:	• SR ORE LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)**
Function description	Clear IDLE line detected Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Clearing this flag is done by a read access to the USARTx_SR register followed by a read access to the USARTx_DR register. • Please also consider that when clearing this flag, other flags as PE, NE, FE, ORE would also be cleared.
Reference Manual to LL API cross reference:	• SR IDLE LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TC

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)**
Function description	Clear Transmission Complete Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR TC LL_USART_ClearFlag_TC

LL_USART_ClearFlag_RXNE

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_RXNE (USART_TypeDef * USARTx)**
Function description	Clear RX Not Empty Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• SR RXNE LL_USART_ClearFlag_RXNE

LL_USART_ClearFlag_LBD

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)**
Function description	Clear LIN Break Detection Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**

Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• SR LBD LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name	**__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)**
Function description	Clear CTS Interrupt Flag.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• SR CTS LL_USART_ClearFlag_nCTS

LL_USART_EnableIT_IDLE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)**
Function description	Enable IDLE Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_RXNE (USART_TypeDef * USARTx)**
Function description	Enable RX Not Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TC

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TC**

(USART_TypeDef * USARTx)

Function description	Enable Transmission Complete Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_TXE (USART_TypeDef * USARTx)**
Function description	Enable TX Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_USART_EnableIT_TXE

LL_USART_EnableIT_PE

Function name	**__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)**
Function description	Enable Parity Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_LBD

Function name	**__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)**
Function description	Enable LIN Break Detection Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name	**__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)**
Function description	Enable Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_SR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_SR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name	**__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)**
Function description	Enable CTS Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_DisableIT_IDLE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)**
Function description	Disable IDLE Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_RXNE (USART_TypeDef * USARTx)**
Function description	Disable RX Not Empty Interrupt.

Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TC

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)**
Function description	Disable Transmission Complete Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_TXE (USART_TypeDef * USARTx)**
Function description	Disable TX Empty Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_USART_DisableIT_TXE

LL_USART_DisableIT_PE

Function name	**__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)**
Function description	Disable Parity Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_LBD

Function name	**__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)**
Function description	Disable LIN Break Detection Interrupt.
Parameters	• **USARTx:** USART Instance

Return values	• **None:**
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name	**__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)**
Function description	Disable Error Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_SR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_SR register.
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name	**__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)**
Function description	Disable CTS Interrupt.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_DisableIT_CTS

LL_USART_IsEnabledIT_IDLE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (USART_TypeDef * USARTx)**
Function description	Check if the USART IDLE Interrupt source is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE (USART_TypeDef * USARTx)**
Function description	Check if the USART RX Not Empty Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 RXNEIE LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TC

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (USART_TypeDef * USARTx)**
Function description	Check if the USART Transmission Complete Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE (USART_TypeDef * USARTx)**
Function description	Check if the USART TX Empty Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR1 TXEIE LL_USART_IsEnabledIT_TXE

LL_USART_IsEnabledIT_PE

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (USART_TypeDef * USARTx)**
Function description	Check if the USART Parity Error Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:	• CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_LBD

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (USART_TypeDef * USARTx)**
Function description	Check if the USART LIN Break Detection Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (USART_TypeDef * USARTx)**
Function description	Check if the USART Error Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (USART_TypeDef * USARTx)**
Function description	Check if the USART CTS Interrupt is enabled or disabled.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Notes	• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.
Reference Manual to LL API cross reference:	• CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_EnableDMAReq_RX

Function name	**__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Enable DMA Mode for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name	**__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Disable DMA Mode for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (USART_TypeDef * USARTx)**
Function description	Check if DMA Mode is enabled for reception.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name	**__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Enable DMA Mode for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name	**__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Disable DMA Mode for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name	**__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (USART_TypeDef * USARTx)**
Function description	Check if DMA Mode is enabled for transmission.
Parameters	• **USARTx:** USART Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_DMA_GetRegAddr

Function name	**__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (USART_TypeDef * USARTx)**
Function description	Get the data register address used for DMA transfer.
Parameters	• **USARTx:** USART Instance
Return values	• **Address:** of data register
Notes	• Address of Data Register is valid for both Transmit and Receive transfers.
Reference Manual to LL API cross reference:	• DR DR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name	**__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (USART_TypeDef * USARTx)**
Function description	Read Receiver Data register (Receive Data value, 8 bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0xFF
Reference Manual to LL API cross reference:	• DR DR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name	**__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (USART_TypeDef * USARTx)**
Function description	Read Receiver Data register (Receive Data value, 9 bits)
Parameters	• **USARTx:** USART Instance
Return values	• **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Reference Manual to LL API cross reference:	• DR DR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name	**__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	• **USARTx:** USART Instance • **Value:** between Min_Data=0x00 and Max_Data=0xFF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name	**__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)**
Function description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	• **USARTx:** USART Instance • **Value:** between Min_Data=0x00 and Max_Data=0x1FF
Return values	• **None:**
Reference Manual to LL API cross reference:	• DR DR LL_USART_TransmitData9

LL_USART_RequestBreakSending

Function name	**__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)**
Function description	Request Break sending.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 SBK LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name	**__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)**
Function description	Put USART in Mute mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RWU LL_USART_RequestEnterMuteMode

LL_USART_RequestExitMuteMode

Function name	**__STATIC_INLINE void LL_USART_RequestExitMuteMode (USART_TypeDef * USARTx)**
Function description	Put USART in Active mode.
Parameters	• **USARTx:** USART Instance
Return values	• **None:**
Reference Manual to LL API cross reference:	• CR1 RWU LL_USART_RequestExitMuteMode

LL_USART_DeInit

Function name	**ErrorStatus LL_USART_DeInit (USART_TypeDef * USARTx)**
Function description	De-initialize USART registers (Registers restored to their default values).
Parameters	• **USARTx:** USART Instance
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers are de-initialized – ERROR: USART registers are not de-initialized

LL_USART_Init

Function name	**ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, LL_USART_InitTypeDef * USART_InitStruct)**
Function description	Initialize USART registers according to the specified parameters in USART_InitStruct.
Parameters	• **USARTx:** USART Instance • **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.
Return values	• **An:** ErrorStatus enumeration value: – SUCCESS: USART registers are initialized according to USART_InitStruct content – ERROR: Problem occurred during USART Registers initialization

Notes
- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name **void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)**

Function description Set each LL_USART_InitTypeDef field to default value.

Parameters
- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.

Return values
- **None:**

LL_USART_ClockInit

Function name **ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, LL_USART_ClockInitTypeDef * USART_ClockInitStruct)**

Function description Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters
- **USARTx:** USART Instance
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.

Return values
- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes
- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name **void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)**

Function description Set each field of a LL_USART_ClockInitTypeDef type structure to default value.

Parameters
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.

Return values
- **None:**

63.3 USART Firmware driver defines

63.3.1 USART

Clock Signal

LL_USART_CLOCK_DISABLE	Clock signal not provided
LL_USART_CLOCK_ENABLE	Clock signal provided

Datawidth

LL_USART_DATAWIDTH_8B	8 bits word length : Start bit, 8 data bits, n stop bits
LL_USART_DATAWIDTH_9B	9 bits word length : Start bit, 9 data bits, n stop bits

Communication Direction

LL_USART_DIRECTION_NONE	Transmitter and Receiver are disabled
LL_USART_DIRECTION_RX	Transmitter is disabled and Receiver is enabled
LL_USART_DIRECTION_TX	Transmitter is enabled and Receiver is disabled
LL_USART_DIRECTION_TX_RX	Transmitter and Receiver are enabled

Get Flags Defines

LL_USART_SR_PE	Parity error flag
LL_USART_SR_FE	Framing error flag
LL_USART_SR_NE	Noise detected flag
LL_USART_SR_ORE	Overrun error flag
LL_USART_SR_IDLE	Idle line detected flag
LL_USART_SR_RXNE	Read data register not empty flag
LL_USART_SR_TC	Transmission complete flag
LL_USART_SR_TXE	Transmit data register empty flag
LL_USART_SR_LBD	LIN break detection flag
LL_USART_SR_CTS	CTS flag

Hardware Control

LL_USART_HWCONTROL_NONE	CTS and RTS hardware flow control disabled
LL_USART_HWCONTROL_RTS	RTS output enabled, data is only requested when there is space in the receive buffer
LL_USART_HWCONTROL_CTS	CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
LL_USART_HWCONTROL_RTS_CTS	CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL	IrDA normal power mode
LL_USART_IRDA_POWER_LOW	IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE	IDLE interrupt enable

LL_USART_CR1_RXNEIE	Read data register not empty interrupt enable
LL_USART_CR1_TCIE	Transmission complete interrupt enable
LL_USART_CR1_TXEIE	Transmit data register empty interrupt enable
LL_USART_CR1_PEIE	Parity error
LL_USART_CR2_LBDIE	LIN break detection interrupt enable
LL_USART_CR3_EIE	Error interrupt enable
LL_USART_CR3_CTSIE	CTS interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT	The clock pulse of the last data bit is not output to the SCLK pin
LL_USART_LASTCLKPULSE_OUTPUT	The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B	10-bit break detection method selected
LL_USART_LINBREAK_DETECT_11B	11-bit break detection method selected

Parity Control

LL_USART_PARITY_NONE	Parity control disabled
LL_USART_PARITY_EVEN	Parity control enabled and Even Parity is selected
LL_USART_PARITY_ODD	Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE	The first clock transition is the first data capture edge
LL_USART_PHASE_2EDGE	The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW	Steady low value on SCLK pin outside transmission window
LL_USART_POLARITY_HIGH	Steady high value on SCLK pin outside transmission window

Stop Bits

LL_USART_STOPBITS_0_5	0.5 stop bit
LL_USART_STOPBITS_1	1 stop bit
LL_USART_STOPBITS_1_5	1.5 stop bits
LL_USART_STOPBITS_2	2 stop bits

Wakeup

LL_USART_WAKEUP_IDLELINE	USART wake up from Mute mode on Idle Line
LL_USART_WAKEUP_ADDRESSMARK	USART wake up from Mute mode on Address Mark

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8_100	**Description:**

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIVMANT_SAMPLING8

__LL_USART_DIVFRAQ_SAMPLING8

__LL_USART_DIV_SAMPLING8

__LL_USART_DIV_SAMPLING16_100

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

__LL_USART_DIVMANT_SAMPLING16

__LL_USART_DIVFRAQ_SAMPLING16

__LL_USART_DIV_SAMPLING16

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

64 LL UTILS Generic Driver

64.1 UTILS Firmware driver registers structures

64.1.1 LL_UTILS_PLLInitTypeDef

Data Fields

- ***uint32_t PLLMul***
- ***uint32_t Prediv***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLMul***
  Multiplication factor for PLL VCO input clock. This parameter can be a value of ***RCC_LL_EC_PLL_MUL***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::Prediv***
  Division factor for HSE used as PLL clock source. This parameter can be a value of ***RCC_LL_EC_PREDIV_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.

64.1.2 LL_UTILS_ClkInitTypeDef

Data Fields

- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.

64.2 UTILS Firmware driver API description

64.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is RCC_MAX_FREQUENCY Hz.

This section contains the following APIs:

- ***LL_SetSystemCoreClock()***

- ***LL_PLL_ConfigSystemClock_HSI()***
- ***LL_PLL_ConfigSystemClock_HSE()***

64.2.2 Detailed description of functions

LL_GetUID_Word0

Function name **__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)**

Function description Get Word0 of the unique device identifier (UID based on 96 bits)

Return values
- **UID[31:0]:**

LL_GetUID_Word1

Function name **__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)**

Function description Get Word1 of the unique device identifier (UID based on 96 bits)

Return values
- **UID[63:32]:**

LL_GetUID_Word2

Function name **__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)**

Function description Get Word2 of the unique device identifier (UID based on 96 bits)

Return values
- **UID[95:64]:**

LL_GetFlashSize

Function name **__STATIC_INLINE uint32_t LL_GetFlashSize (void)**

Function description Get Flash memory size.

Return values
- **FLASH_SIZE[15:0]:** Flash memory size

Notes
- This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_InitTick

Function name **__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)**

Function description This function configures the Cortex-M SysTick source of the time base.

Parameters
- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
- **Ticks:** Number of ticks

Return values
- **None:**

Notes
- When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name	**void LL_Init1msTick (uint32_t HCLKFrequency)**
Function description	This function configures the Cortex-M SysTick source to have 1ms time base.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz
Return values	• **None:**
Notes	• When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service. • HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name	**void LL_mDelay (uint32_t Delay)**
Function description	This function provides accurate delay (in milliseconds) based on SysTick counter flag.
Parameters	• **Delay:** specifies the delay time length, in milliseconds.
Return values	• **None:**
Notes	• When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service. • To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name	**void LL_SetSystemCoreClock (uint32_t HCLKFrequency)**
Function description	This function sets directly SystemCoreClock CMSIS variable.
Parameters	• **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
Return values	• **None:**
Notes	• Variable can be calculated also through SystemCoreClockUpdate function.

LL_PLL_ConfigSystemClock_HSI

Function name	**ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)**
Function description	This function configures system clock with HSI as clock source of the PLL.
Parameters	• **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL. • **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the

BUS prescalers.

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: Max frequency configuration done
  – ERROR: Max frequency configuration not done

Notes
- The application need to ensure that PLL is disabled.
- Function is based on the following formula: PLL output frequency = ((HSI frequency / PREDIV) * PLLMUL)PREDIV: Set to 2 for few devicesPLLMUL: The application software must set correctly the PLL multiplication factor to not exceed 72MHz
- FLASH latency can be modified through this function.

LL_PLL_ConfigSystemClock_HSE

Function name **ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)**

Function description This function configures system clock with HSE as clock source of the PLL.

Parameters
- **HSEFrequency:** Value between Min_Data = RCC_HSE_MIN and Max_Data = RCC_HSE_MAX
- **HSEBypass:** This parameter can be one of the following values:
  – LL_UTILS_HSEBYPASS_ON
  – LL_UTILS_HSEBYPASS_OFF
- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values
- **An:** ErrorStatus enumeration value:
  – SUCCESS: Max frequency configuration done
  – ERROR: Max frequency configuration not done

Notes
- The application need to ensure that PLL is disabled.
- Function is based on the following formula: PLL output frequency = ((HSI frequency / PREDIV) * PLLMUL)PREDIV: Set to 2 for few devicesPLLMUL: The application software must set correctly the PLL multiplication factor to not exceed UTILS_PLL_OUTPUT_MAX
- FLASH latency can be modified through this function.

64.3 UTILS Firmware driver defines

64.3.1 UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON	HSE Bypass is enabled

65 LL WWDG Generic Driver

65.1 WWDG Firmware driver API description

65.1.1 Detailed description of functions

LL_WWDG_Enable

Function name	**__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)**
Function description	Enable Window Watchdog.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **None:**
Notes	• It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.
Reference Manual to LL API cross reference:	• CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name	**__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)**
Function description	Checks if Window Watchdog is enabled.
Parameters	• **WWDGx:** WWDG Instance
Return values	• **State:** of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name	**__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)**
Function description	Set the Watchdog counter value to provided value (7-bits T[6:0])
Parameters	• **WWDGx:** WWDG Instance • **Counter:** 0..0x7F (7 bit counter value)
Return values	• **None:**
Notes	• When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6

becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)

Reference Manual to LL API cross reference:
- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name: **__STATIC_INLINE uint32_t LL_WWDG_GetCounter (WWDG_TypeDef * WWDGx)**

Function description: Return current Watchdog Counter Value (7 bits counter value)

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **7:** bit Watchdog Counter value

Reference Manual to LL API cross reference:
- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name: **__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)**

Function description: Set the time base of the prescaler (WDGTB).

Parameters:
- **WWDGx:** WWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8

Return values:
- **None:**

Notes:
- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles

Reference Manual to LL API cross reference:
- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name: **__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)**

Function description: Return current Watchdog Prescaler Value.

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4

– LL_WWDG_PRESCALER_8

Reference Manual to LL API cross reference:
- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name: **__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)**

Function description: Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters:
- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values:
- **None:**

Notes:
- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:
- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name: **__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)**

Function description: Return current Watchdog Window Value (7 bits value)

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:
- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name: **__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)**

Function description: Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **State:** of bit (1 or 0).

Notes:
- This bit is set by hardware when the counter has reached the

value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:
- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name: **__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)**

Function description: Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **None:**

Reference Manual to LL API cross reference:
- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name: **__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)**

Function description: Enable the Early Wakeup Interrupt.

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **None:**

Notes:
- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:
- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name: **__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)**

Function description: Check if Early Wakeup Interrupt is enabled.

Parameters:
- **WWDGx:** WWDG Instance

Return values:
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

65.2 WWDG Firmware driver defines

65.2.1 WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1 WWDG counter clock = (PCLK1/4096)/1

LL_WWDG_PRESCALER_2 WWDG counter clock = (PCLK1/4096)/2

LL_WWDG_PRESCALER_4 WWDG counter clock = (PCLK1/4096)/4

LL_WWDG_PRESCALER_8 WWDG counter clock = (PCLK1/4096)/8

Common Write and read registers macros

LL_WWDG_WriteReg **Description:**

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_WWDG_ReadReg **Description:**

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

66 Correspondence between API registers and API low-layer driver functions

66.1 ADC

Table 25: Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
CR1	AWD1CH	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
	AWD1EN	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
	AWD1SGL	*LL_ADC_GetAnalogWDMonitChannels*
		LL_ADC_SetAnalogWDMonitChannels
	AWDIE	*LL_ADC_EnableIT_AWD1*
	DISCEN	*LL_ADC_INJ_SetSequencerDiscont*
		LL_ADC_REG_GetSequencerDiscont
		LL_ADC_REG_SetSequencerDiscont
	DISCNUM	*LL_ADC_REG_GetSequencerDiscont*
		LL_ADC_REG_SetSequencerDiscont
	DUALMOD	*LL_ADC_GetMultimode*
		LL_ADC_SetMultimode
	EOCIE	*LL_ADC_DisableIT_EOS*
		LL_ADC_EnableIT_EOS
		LL_ADC_IsEnabledIT_EOS
	JAUTO	*LL_ADC_INJ_GetTrigAuto*
		LL_ADC_INJ_SetTrigAuto
	JEOCIE	*LL_ADC_EnableIT_JEOS*
	SCAN	*LL_ADC_GetSequencersScanMode*
		LL_ADC_SetSequencersScanMode
CR2	ADON	*LL_ADC_Disable*
		LL_ADC_Enable
		LL_ADC_IsEnabled
	ALIGN	*LL_ADC_SetDataAlignment*
	CAL	*LL_ADC_IsCalibrationOnGoing*
		LL_ADC_StartCalibration
	CONT	*LL_ADC_REG_GetContinuousMode*
		LL_ADC_REG_SetContinuousMode

Register	Field	Function
	DMA	*LL_ADC_REG_GetDMATransfer*
		LL_ADC_REG_SetDMATransfer
	EXTEN	*LL_ADC_REG_StartConversionExtTrig*
	EXTSEL	*LL_ADC_REG_GetTriggerSource*
		LL_ADC_REG_IsTriggerSourceSWStart
		LL_ADC_REG_SetTriggerSource
		LL_ADC_REG_StopConversionExtTrig
	JEXTEN	*LL_ADC_INJ_StartConversionExtTrig*
	JEXTSEL	*LL_ADC_INJ_GetTriggerSource*
		LL_ADC_INJ_IsTriggerSourceSWStart
		LL_ADC_INJ_SetTriggerSource
		LL_ADC_INJ_StopConversionExtTrig
	JSWSTART	*LL_ADC_INJ_StartConversionSWStart*
	SWSTART	*LL_ADC_REG_StartConversionSWStart*
	TSVREFE	*LL_ADC_GetCommonPathInternalCh*
		LL_ADC_SetCommonPathInternalCh
DR	ADC2DATA	*LL_ADC_REG_ReadMultiConversionData32*
	DATA	*LL_ADC_DMA_GetRegAddr*
		LL_ADC_REG_ReadMultiConversionData32
	RDATA	*LL_ADC_REG_ReadConversionData12*
		LL_ADC_REG_ReadConversionData32
HTR	HT	*LL_ADC_GetAnalogWDThresholds*
		LL_ADC_SetAnalogWDThresholds
JDR1	JDATA	*LL_ADC_INJ_ReadConversionData12*
		LL_ADC_INJ_ReadConversionData32
JDR2	JDATA	*LL_ADC_INJ_ReadConversionData12*
		LL_ADC_INJ_ReadConversionData32
JDR3	JDATA	*LL_ADC_INJ_ReadConversionData12*
		LL_ADC_INJ_ReadConversionData32
JDR4	JDATA	*LL_ADC_INJ_ReadConversionData12*
		LL_ADC_INJ_ReadConversionData32
JOFR1	JOFFSET1	*LL_ADC_INJ_GetOffset*
		LL_ADC_INJ_SetOffset
JOFR2	JOFFSET2	*LL_ADC_INJ_GetOffset*
		LL_ADC_INJ_SetOffset
JOFR3	JOFFSET3	*LL_ADC_INJ_GetOffset*

Register	Field	Function
		LL_ADC_INJ_SetOffset
JOFR4	JOFFSET4	*LL_ADC_INJ_GetOffset*
		LL_ADC_INJ_SetOffset
JSQR	JL	*LL_ADC_INJ_GetSequencerLength*
		LL_ADC_INJ_SetSequencerLength
	JSQ1	*LL_ADC_INJ_SetSequencerRanks*
	JSQ2	*LL_ADC_INJ_SetSequencerRanks*
	JSQ3	*LL_ADC_INJ_SetSequencerRanks*
	JSQ4	*LL_ADC_INJ_SetSequencerRanks*
LTR	LT	*LL_ADC_GetAnalogWDThresholds*
		LL_ADC_SetAnalogWDThresholds
SMPR1	SMP10	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP11	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP12	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP13	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP14	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP15	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP16	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP17	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
SMPR2	SMP0	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP1	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP2	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP3	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP4	*LL_ADC_GetChannelSamplingTime*

Register	Field	Function
		LL_ADC_SetChannelSamplingTime
	SMP5	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP6	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP7	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP8	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
	SMP9	*LL_ADC_GetChannelSamplingTime*
		LL_ADC_SetChannelSamplingTime
SQR1	L	*LL_ADC_REG_SetSequencerLength*
	SQ13	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ14	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ15	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ16	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
SQR2	SQ10	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ11	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ12	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ7	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ8	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ9	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
SQR3	SQ1	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ2	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks

Register	Field	Function
	SQ3	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ4	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ5	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
	SQ6	*LL_ADC_REG_GetSequencerRanks*
		LL_ADC_REG_SetSequencerRanks
SR	AWD	*LL_ADC_ClearFlag_AWD1*
		LL_ADC_IsActiveFlag_AWD1
		LL_ADC_IsActiveFlag_MST_AWD1
		LL_ADC_IsActiveFlag_SLV_AWD1
	EOC	*LL_ADC_ClearFlag_EOS*
		LL_ADC_IsActiveFlag_EOS
		LL_ADC_IsActiveFlag_MST_EOS
		LL_ADC_IsActiveFlag_SLV_EOS
	JEOC	*LL_ADC_ClearFlag_JEOS*
		LL_ADC_IsActiveFlag_JEOS
		LL_ADC_IsActiveFlag_MST_JEOS
		LL_ADC_IsActiveFlag_SLV_JEOS

66.2 BUS

Table 26: Correspondence between BUS registers and BUS low-layer driver functions

Register	Field	Function
AHBENR	CRCEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA1EN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA2EN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	ETHMACEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock

Register	Field	Function
	ETHMACRXEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	ETHMACTXEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FLITFEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FSMCEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	OTGFSEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	SDIOEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	SRAMEN	*LL_AHB1_GRP1_DisableClock*
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
APB1ENR	BKPEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CAN1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CAN2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CECEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	DACEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock

Register	Field	Function
	I2C1EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	PWREN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM12EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM13EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM14EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM4EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM5EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock

Register	Field	Function
	TIM6EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM7EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART4EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART5EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART2EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART3EN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USBEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	WWDGEN	*LL_APB1_GRP1_DisableClock*
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1RSTR	BKPRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	CAN1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	CAN2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	CECRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	DACRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	I2C1RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset

Register	Field	Function
	I2C2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	PWRRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	SPI2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	SPI3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM12RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM13RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM14RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM4RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM5RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM6RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	TIM7RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	UART4RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	UART5RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USART2RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USART3RST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
	USBRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset

Register	Field	Function
	WWDGRST	*LL_APB1_GRP1_ForceReset*
		LL_APB1_GRP1_ReleaseReset
APB2ENR	ADC1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	ADC2EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	ADC3EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	AFIOEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPAEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPBEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPCEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPDEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPEEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPFEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	IOPGEN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI1EN	*LL_APB2_GRP1_DisableClock*

Register	Field	Function
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM10EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM11EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM15EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM16EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM17EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM8EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM9EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USART1EN	*LL_APB2_GRP1_DisableClock*
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
APB2RSTR	ADC1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	ADC2RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	ADC3RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	AFIORST	*LL_APB2_GRP1_ForceReset*

Register	Field	Function
		LL_APB2_GRP1_ReleaseReset
	IOPARST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	IOPBRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	IOPCRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	IOPDRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	IOPERST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	IOPFRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	IOPGRST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	SPI1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM10RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM11RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM15RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM16RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM17RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM8RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	TIM9RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset
	USART1RST	*LL_APB2_GRP1_ForceReset*
		LL_APB2_GRP1_ReleaseReset

66.3 CORTEX

Table 27: Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	*LL_MPU_Disable*
		LL_MPU_Enable
		LL_MPU_IsEnabled
MPU_RASR	AP	*LL_MPU_ConfigRegion*
	B	*LL_MPU_ConfigRegion*
	C	*LL_MPU_ConfigRegion*
	ENABLE	*LL_MPU_DisableRegion*
		LL_MPU_EnableRegion
	S	*LL_MPU_ConfigRegion*
	SIZE	*LL_MPU_ConfigRegion*
	XN	*LL_MPU_ConfigRegion*
MPU_RBAR	ADDR	*LL_MPU_ConfigRegion*
	REGION	*LL_MPU_ConfigRegion*
MPU_RNR	REGION	*LL_MPU_ConfigRegion*
		LL_MPU_DisableRegion
SCB_CPUID	ARCHITECTURE	*LL_CPUID_GetConstant*
	IMPLEMENTER	*LL_CPUID_GetImplementer*
	PARTNO	*LL_CPUID_GetParNo*
	REVISION	*LL_CPUID_GetRevision*
	VARIANT	*LL_CPUID_GetVariant*
SCB_SCR	SEVEONPEND	*LL_LPM_DisableEventOnPend*
		LL_LPM_EnableEventOnPend
	SLEEPDEEP	*LL_LPM_EnableDeepSleep*
		LL_LPM_EnableSleep
	SLEEPONEXIT	*LL_LPM_DisableSleepOnExit*
		LL_LPM_EnableSleepOnExit
SCB_SHCSR	MEMFAULTENA	*LL_HANDLER_DisableFault*
		LL_HANDLER_EnableFault
STK_CTRL	CLKSOURCE	*LL_SYSTICK_GetClkSource*
		LL_SYSTICK_SetClkSource
	COUNTFLAG	*LL_SYSTICK_IsActiveCounterFlag*
	TICKINT	*LL_SYSTICK_DisableIT*
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

66.4 CRC

Table 28: Correspondence between CRC registers and CRC low-layer driver functions

Register	Field	Function
CR	RESET	*LL_CRC_ResetCRCCalculationUnit*
DR	DR	*LL_CRC_FeedData32*
		LL_CRC_ReadData32
IDR	IDR	*LL_CRC_Read_IDR*
		LL_CRC_Write_IDR

66.5 DAC

Table 29: Correspondence between DAC registers and DAC low-layer driver functions

Register	Field	Function
CR	BOFF1	*LL_DAC_GetOutputBuffer*
		LL_DAC_SetOutputBuffer
	BOFF2	*LL_DAC_GetOutputBuffer*
		LL_DAC_SetOutputBuffer
	DMAEN1	*LL_DAC_DisableDMAReq*
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAEN2	*LL_DAC_DisableDMAReq*
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	EN1	*LL_DAC_Disable*
		LL_DAC_Enable
		LL_DAC_IsEnabled
	EN2	*LL_DAC_Disable*
		LL_DAC_Enable
		LL_DAC_IsEnabled
	MAMP1	*LL_DAC_GetWaveNoiseLFSR*
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MAMP2	*LL_DAC_GetWaveNoiseLFSR*
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR

Register	Field	Function
		LL_DAC_SetWaveTriangleAmplitude
	TEN1	*LL_DAC_DisableTrigger*
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TEN2	*LL_DAC_DisableTrigger*
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TSEL1	*LL_DAC_GetTriggerSource*
		LL_DAC_SetTriggerSource
	TSEL2	*LL_DAC_GetTriggerSource*
		LL_DAC_SetTriggerSource
	WAVE1	*LL_DAC_GetWaveAutoGeneration*
		LL_DAC_SetWaveAutoGeneration
	WAVE2	*LL_DAC_GetWaveAutoGeneration*
		LL_DAC_SetWaveAutoGeneration
DHR12L1	DACC1DHR	*LL_DAC_ConvertData12LeftAligned*
		LL_DAC_DMA_GetRegAddr
DHR12L2	DACC2DHR	*LL_DAC_ConvertData12LeftAligned*
		LL_DAC_DMA_GetRegAddr
DHR12LD	DACC1DHR	*LL_DAC_ConvertDualData12LeftAligned*
	DACC2DHR	*LL_DAC_ConvertDualData12LeftAligned*
DHR12R1	DACC1DHR	*LL_DAC_ConvertData12RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR12R2	DACC2DHR	*LL_DAC_ConvertData12RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR12RD	DACC1DHR	*LL_DAC_ConvertDualData12RightAligned*
	DACC2DHR	*LL_DAC_ConvertDualData12RightAligned*
DHR8R1	DACC1DHR	*LL_DAC_ConvertData8RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR8R2	DACC2DHR	*LL_DAC_ConvertData8RightAligned*
		LL_DAC_DMA_GetRegAddr
DHR8RD	DACC1DHR	*LL_DAC_ConvertDualData8RightAligned*
	DACC2DHR	*LL_DAC_ConvertDualData8RightAligned*
DOR1	DACC1DOR	*LL_DAC_RetrieveOutputData*
DOR2	DACC2DOR	*LL_DAC_RetrieveOutputData*
SWTRIGR	SWTRIG1	*LL_DAC_TrigSWConversion*

Register	Field	Function
	SWTRIG2	*LL_DAC_TrigSWConversion*

66.6 DMA

Table 30: Correspondence between DMA registers and DMA low-layer driver functions

Register	Field	Function
CCR	CIRC	*LL_DMA_ConfigTransfer*
		LL_DMA_GetMode
		LL_DMA_SetMode
	DIR	*LL_DMA_ConfigTransfer*
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	EN	*LL_DMA_DisableChannel*
		LL_DMA_EnableChannel
		LL_DMA_IsEnabledChannel
	HTIE	*LL_DMA_DisableIT_HT*
		LL_DMA_EnableIT_HT
		LL_DMA_IsEnabledIT_HT
	MEM2MEM	*LL_DMA_ConfigTransfer*
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	MINC	*LL_DMA_ConfigTransfer*
		LL_DMA_GetMemoryIncMode
		LL_DMA_SetMemoryIncMode
	MSIZE	*LL_DMA_ConfigTransfer*
		LL_DMA_GetMemorySize
		LL_DMA_SetMemorySize
	PINC	*LL_DMA_ConfigTransfer*
		LL_DMA_GetPeriphIncMode
		LL_DMA_SetPeriphIncMode
	PL	*LL_DMA_ConfigTransfer*
		LL_DMA_GetChannelPriorityLevel
		LL_DMA_SetChannelPriorityLevel
	PSIZE	*LL_DMA_ConfigTransfer*
		LL_DMA_GetPeriphSize
		LL_DMA_SetPeriphSize
	TCIE	*LL_DMA_DisableIT_TC*

Register	Field	Function
		LL_DMA_EnableIT_TC
		LL_DMA_IsEnabledIT_TC
	TEIE	*LL_DMA_DisableIT_TE*
		LL_DMA_EnableIT_TE
		LL_DMA_IsEnabledIT_TE
CMAR	MA	*LL_DMA_ConfigAddresses*
		LL_DMA_GetM2MDstAddress
		LL_DMA_GetMemoryAddress
		LL_DMA_SetM2MDstAddress
		LL_DMA_SetMemoryAddress
CNDTR	NDT	*LL_DMA_GetDataLength*
		LL_DMA_SetDataLength
CPAR	PA	*LL_DMA_ConfigAddresses*
		LL_DMA_GetM2MSrcAddress
		LL_DMA_GetPeriphAddress
		LL_DMA_SetM2MSrcAddress
		LL_DMA_SetPeriphAddress
IFCR	CGIF1	*LL_DMA_ClearFlag_GI1*
	CGIF2	*LL_DMA_ClearFlag_GI2*
	CGIF3	*LL_DMA_ClearFlag_GI3*
	CGIF4	*LL_DMA_ClearFlag_GI4*
	CGIF5	*LL_DMA_ClearFlag_GI5*
	CGIF6	*LL_DMA_ClearFlag_GI6*
	CGIF7	*LL_DMA_ClearFlag_GI7*
	CHTIF1	*LL_DMA_ClearFlag_HT1*
	CHTIF2	*LL_DMA_ClearFlag_HT2*
	CHTIF3	*LL_DMA_ClearFlag_HT3*
	CHTIF4	*LL_DMA_ClearFlag_HT4*
	CHTIF5	*LL_DMA_ClearFlag_HT5*
	CHTIF6	*LL_DMA_ClearFlag_HT6*
	CHTIF7	*LL_DMA_ClearFlag_HT7*
	CTCIF1	*LL_DMA_ClearFlag_TC1*
	CTCIF2	*LL_DMA_ClearFlag_TC2*
	CTCIF3	*LL_DMA_ClearFlag_TC3*
	CTCIF4	*LL_DMA_ClearFlag_TC4*
	CTCIF5	*LL_DMA_ClearFlag_TC5*

Register	Field	Function
	CTCIF6	*LL_DMA_ClearFlag_TC6*
	CTCIF7	*LL_DMA_ClearFlag_TC7*
	CTEIF1	*LL_DMA_ClearFlag_TE1*
	CTEIF2	*LL_DMA_ClearFlag_TE2*
	CTEIF3	*LL_DMA_ClearFlag_TE3*
	CTEIF4	*LL_DMA_ClearFlag_TE4*
	CTEIF5	*LL_DMA_ClearFlag_TE5*
	CTEIF6	*LL_DMA_ClearFlag_TE6*
	CTEIF7	*LL_DMA_ClearFlag_TE7*
ISR	GIF1	*LL_DMA_IsActiveFlag_GI1*
	GIF2	*LL_DMA_IsActiveFlag_GI2*
	GIF3	*LL_DMA_IsActiveFlag_GI3*
	GIF4	*LL_DMA_IsActiveFlag_GI4*
	GIF5	*LL_DMA_IsActiveFlag_GI5*
	GIF6	*LL_DMA_IsActiveFlag_GI6*
	GIF7	*LL_DMA_IsActiveFlag_GI7*
	HTIF1	*LL_DMA_IsActiveFlag_HT1*
	HTIF2	*LL_DMA_IsActiveFlag_HT2*
	HTIF3	*LL_DMA_IsActiveFlag_HT3*
	HTIF4	*LL_DMA_IsActiveFlag_HT4*
	HTIF5	*LL_DMA_IsActiveFlag_HT5*
	HTIF6	*LL_DMA_IsActiveFlag_HT6*
	HTIF7	*LL_DMA_IsActiveFlag_HT7*
	TCIF1	*LL_DMA_IsActiveFlag_TC1*
	TCIF2	*LL_DMA_IsActiveFlag_TC2*
	TCIF3	*LL_DMA_IsActiveFlag_TC3*
	TCIF4	*LL_DMA_IsActiveFlag_TC4*
	TCIF5	*LL_DMA_IsActiveFlag_TC5*
	TCIF6	*LL_DMA_IsActiveFlag_TC6*
	TCIF7	*LL_DMA_IsActiveFlag_TC7*
	TEIF1	*LL_DMA_IsActiveFlag_TE1*
	TEIF2	*LL_DMA_IsActiveFlag_TE2*
	TEIF3	*LL_DMA_IsActiveFlag_TE3*
	TEIF4	*LL_DMA_IsActiveFlag_TE4*
	TEIF5	*LL_DMA_IsActiveFlag_TE5*
	TEIF6	*LL_DMA_IsActiveFlag_TE6*

Register	Field	Function
	TEIF7	*LL_DMA_IsActiveFlag_TE7*

66.7 EXTI

Table 31: Correspondence between EXTI registers and EXTI low-layer driver functions

Register	Field	Function
EMR	EMx	*LL_EXTI_DisableEvent_0_31*
		LL_EXTI_EnableEvent_0_31
		LL_EXTI_IsEnabledEvent_0_31
FTSR	FTx	*LL_EXTI_DisableFallingTrig_0_31*
		LL_EXTI_EnableFallingTrig_0_31
		LL_EXTI_IsEnabledFallingTrig_0_31
IMR	IMx	*LL_EXTI_DisableIT_0_31*
		LL_EXTI_EnableIT_0_31
		LL_EXTI_IsEnabledIT_0_31
PR	PIFx	*LL_EXTI_ClearFlag_0_31*
		LL_EXTI_IsActiveFlag_0_31
		LL_EXTI_ReadFlag_0_31
RTSR	RTx	*LL_EXTI_DisableRisingTrig_0_31*
		LL_EXTI_EnableRisingTrig_0_31
		LL_EXTI_IsEnabledRisingTrig_0_31
SWIER	SWIx	*LL_EXTI_GenerateSWI_0_31*

66.8 GPIO

Table 32: Correspondence between GPIO registers and GPIO low-layer driver functions

Register	Field	Function
BRR	BRy	*LL_GPIO_ResetOutputPin*
BSRR	BSy	*LL_GPIO_SetOutputPin*
CRH	CNFy	*LL_GPIO_GetPinMode*
		LL_GPIO_SetPinMode
	MODEy	*LL_GPIO_GetPinMode*
		LL_GPIO_GetPinOutputType
		LL_GPIO_GetPinSpeed
		LL_GPIO_SetPinMode
		LL_GPIO_SetPinOutputType
		LL_GPIO_SetPinSpeed

Register	Field	Function
CRL	CNFy	*LL_GPIO_GetPinMode*
		LL_GPIO_SetPinMode
	MODEy	*LL_GPIO_GetPinMode*
		LL_GPIO_GetPinOutputType
		LL_GPIO_GetPinSpeed
		LL_GPIO_SetPinMode
		LL_GPIO_SetPinOutputType
		LL_GPIO_SetPinSpeed
IDR	IDy	*LL_GPIO_IsInputPinSet*
		LL_GPIO_ReadInputPort
LCKR	LCKK	*LL_GPIO_IsAnyPinLocked*
		LL_GPIO_LockPin
	LCKy	*LL_GPIO_IsPinLocked*
ODR	ODR	*LL_GPIO_GetPinPull*
		LL_GPIO_SetPinPull
	ODy	*LL_GPIO_IsOutputPinSet*
		LL_GPIO_ReadOutputPort
		LL_GPIO_TogglePin
		LL_GPIO_WriteOutputPort

66.9 I2C

Table 33: Correspondence between I2C registers and I2C low-layer driver functions

Register	Field	Function
CCR	CCR	*LL_I2C_ConfigSpeed*
		LL_I2C_GetClockPeriod
		LL_I2C_SetClockPeriod
	DUTY	*LL_I2C_ConfigSpeed*
		LL_I2C_GetDutyCycle
		LL_I2C_SetDutyCycle
	FS	*LL_I2C_ConfigSpeed*
		LL_I2C_GetClockSpeedMode
		LL_I2C_SetClockSpeedMode
CR1	ACK	*LL_I2C_AcknowledgeNextData*
	ALERT	*LL_I2C_DisableSMBusAlert*
		LL_I2C_EnableSMBusAlert
		LL_I2C_IsEnabledSMBusAlert

Register	Field	Function
	ENARP	*LL_I2C_GetMode*
		LL_I2C_SetMode
	ENGC	*LL_I2C_DisableGeneralCall*
		LL_I2C_EnableGeneralCall
		LL_I2C_IsEnabledGeneralCall
	ENPEC	*LL_I2C_DisableSMBusPEC*
		LL_I2C_EnableSMBusPEC
		LL_I2C_IsEnabledSMBusPEC
	NOSTRETCH	*LL_I2C_DisableClockStretching*
		LL_I2C_EnableClockStretching
		LL_I2C_IsEnabledClockStretching
	PE	*LL_I2C_ClearFlag_STOP*
		LL_I2C_Disable
		LL_I2C_Enable
		LL_I2C_IsEnabled
	PEC	*LL_I2C_DisableSMBusPECCompare*
		LL_I2C_EnableSMBusPECCompare
		LL_I2C_IsEnabledSMBusPECCompare
	POS	*LL_I2C_DisableBitPOS*
		LL_I2C_EnableBitPOS
		LL_I2C_IsEnabledBitPOS
	SMBTYPE	*LL_I2C_GetMode*
		LL_I2C_SetMode
	SMBUS	*LL_I2C_GetMode*
		LL_I2C_SetMode
	START	*LL_I2C_GenerateStartCondition*
	STOP	*LL_I2C_GenerateStopCondition*
	SWRST	*LL_I2C_DisableReset*
		LL_I2C_EnableReset
		LL_I2C_IsResetEnabled
CR2	DMAEN	*LL_I2C_DisableDMAReq_RX*
		LL_I2C_DisableDMAReq_TX
		LL_I2C_EnableDMAReq_RX
		LL_I2C_EnableDMAReq_TX
		LL_I2C_IsEnabledDMAReq_RX
		LL_I2C_IsEnabledDMAReq_TX

Register	Field	Function
	FREQ	*LL_I2C_ConfigSpeed*
		LL_I2C_GetPeriphClock
		LL_I2C_SetPeriphClock
	ITBUFEN	*LL_I2C_DisableIT_BUF*
		LL_I2C_DisableIT_RX
		LL_I2C_DisableIT_TX
		LL_I2C_EnableIT_BUF
		LL_I2C_EnableIT_RX
		LL_I2C_EnableIT_TX
		LL_I2C_IsEnabledIT_BUF
		LL_I2C_IsEnabledIT_RX
		LL_I2C_IsEnabledIT_TX
	ITERREN	*LL_I2C_DisableIT_ERR*
		LL_I2C_EnableIT_ERR
		LL_I2C_IsEnabledIT_ERR
	ITEVTEN	*LL_I2C_DisableIT_EVT*
		LL_I2C_DisableIT_RX
		LL_I2C_DisableIT_TX
		LL_I2C_EnableIT_EVT
		LL_I2C_EnableIT_RX
		LL_I2C_EnableIT_TX
		LL_I2C_IsEnabledIT_EVT
		LL_I2C_IsEnabledIT_RX
		LL_I2C_IsEnabledIT_TX
	LAST	*LL_I2C_DisableLastDMA*
		LL_I2C_EnableLastDMA
		LL_I2C_IsEnabledLastDMA
DR	DR	*LL_I2C_DMA_GetRegAddr*
		LL_I2C_ReceiveData8
		LL_I2C_TransmitData8
OAR1	ADD0	*LL_I2C_SetOwnAddress1*
	ADD1_7	*LL_I2C_SetOwnAddress1*
	ADD8_9	*LL_I2C_SetOwnAddress1*
	ADDMODE	*LL_I2C_SetOwnAddress1*
OAR2	ADD2	*LL_I2C_SetOwnAddress2*
	ENDUAL	*LL_I2C_DisableOwnAddress2*

Register	Field	Function
		LL_I2C_EnableOwnAddress2
		LL_I2C_IsEnabledOwnAddress2
SR1	ADD10	*LL_I2C_IsActiveFlag_ADD10*
	ADDR	*LL_I2C_ClearFlag_ADDR*
		LL_I2C_IsActiveFlag_ADDR
	AF	*LL_I2C_ClearFlag_AF*
		LL_I2C_IsActiveFlag_AF
	ARLO	*LL_I2C_ClearFlag_ARLO*
		LL_I2C_IsActiveFlag_ARLO
	BERR	*LL_I2C_ClearFlag_BERR*
		LL_I2C_IsActiveFlag_BERR
	BTF	*LL_I2C_IsActiveFlag_BTF*
	OVR	*LL_I2C_ClearFlag_OVR*
		LL_I2C_IsActiveFlag_OVR
	PECERR	*LL_I2C_ClearSMBusFlag_PECERR*
		LL_I2C_IsActiveSMBusFlag_PECERR
	RXNE	*LL_I2C_IsActiveFlag_RXNE*
	SB	*LL_I2C_IsActiveFlag_SB*
	SMBALERT	*LL_I2C_ClearSMBusFlag_ALERT*
		LL_I2C_IsActiveSMBusFlag_ALERT
	STOPF	*LL_I2C_ClearFlag_STOP*
		LL_I2C_IsActiveFlag_STOP
	TIMEOUT	*LL_I2C_ClearSMBusFlag_TIMEOUT*
		LL_I2C_IsActiveSMBusFlag_TIMEOUT
	TXE	*LL_I2C_IsActiveFlag_TXE*
SR2	BUSY	*LL_I2C_IsActiveFlag_BUSY*
	DUALF	*LL_I2C_IsActiveFlag_DUAL*
	GENCALL	*LL_I2C_IsActiveFlag_GENCALL*
	MSL	*LL_I2C_IsActiveFlag_MSL*
	PEC	*LL_I2C_GetSMBusPEC*
	SMBDEFAULT	*LL_I2C_IsActiveSMBusFlag_SMBDEFAULT*
	SMBHOST	*LL_I2C_IsActiveSMBusFlag_SMBHOST*
	TRA	*LL_I2C_GetTransferDirection*
TRISE	TRISE	*LL_I2C_ConfigSpeed*
		LL_I2C_GetRiseTime
		LL_I2C_SetRiseTime

66.10 I2S

Table 34: Correspondence between I2S registers and I2S low-layer driver functions

Register	Field	Function
CR2	ERRIE	*LL_I2S_DisableIT_ERR*
		LL_I2S_EnableIT_ERR
		LL_I2S_IsEnabledIT_ERR
	RXDMAEN	*LL_I2S_DisableDMAReq_RX*
		LL_I2S_EnableDMAReq_RX
		LL_I2S_IsEnabledDMAReq_RX
	RXNEIE	*LL_I2S_DisableIT_RXNE*
		LL_I2S_EnableIT_RXNE
		LL_I2S_IsEnabledIT_RXNE
	TXDMAEN	*LL_I2S_DisableDMAReq_TX*
		LL_I2S_EnableDMAReq_TX
		LL_I2S_IsEnabledDMAReq_TX
	TXEIE	*LL_I2S_DisableIT_TXE*
		LL_I2S_EnableIT_TXE
		LL_I2S_IsEnabledIT_TXE
DR	DR	*LL_I2S_ReceiveData16*
		LL_I2S_TransmitData16
I2SCFGR	CHLEN	*LL_I2S_GetDataFormat*
		LL_I2S_SetDataFormat
	CKPOL	*LL_I2S_GetClockPolarity*
		LL_I2S_SetClockPolarity
	DATLEN	*LL_I2S_GetDataFormat*
		LL_I2S_SetDataFormat
	I2SCFG	*LL_I2S_GetTransferMode*
		LL_I2S_SetTransferMode
	I2SE	*LL_I2S_Disable*
		LL_I2S_Enable
		LL_I2S_IsEnabled
	I2SMOD	*LL_I2S_Enable*
	I2SSTD	*LL_I2S_GetStandard*
		LL_I2S_SetStandard
	PCMSYNC	*LL_I2S_GetStandard*
		LL_I2S_SetStandard

Register	Field	Function
I2SPR	I2SDIV	*LL_I2S_GetPrescalerLinear*
		LL_I2S_SetPrescalerLinear
	MCKOE	*LL_I2S_DisableMasterClock*
		LL_I2S_EnableMasterClock
		LL_I2S_IsEnabledMasterClock
	ODD	*LL_I2S_GetPrescalerParity*
		LL_I2S_SetPrescalerParity
SR	BSY	*LL_I2S_IsActiveFlag_BSY*
	CHSIDE	*LL_I2S_IsActiveFlag_CHSIDE*
	FRE	*LL_I2S_ClearFlag_FRE*
		LL_I2S_IsActiveFlag_FRE
	OVR	*LL_I2S_ClearFlag_OVR*
		LL_I2S_IsActiveFlag_OVR
	RXNE	*LL_I2S_IsActiveFlag_RXNE*
	TXE	*LL_I2S_IsActiveFlag_TXE*
	UDR	*LL_I2S_ClearFlag_UDR*
		LL_I2S_IsActiveFlag_UDR

66.11 IWDG

Table 35: Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
KR	KEY	*LL_IWDG_DisableWriteAccess*
		LL_IWDG_Enable
		LL_IWDG_EnableWriteAccess
		LL_IWDG_ReloadCounter
PR	PR	*LL_IWDG_GetPrescaler*
		LL_IWDG_SetPrescaler
RLR	RL	*LL_IWDG_GetReloadCounter*
		LL_IWDG_SetReloadCounter
SR	PVU	*LL_IWDG_IsActiveFlag_PVU*
		LL_IWDG_IsReady
	RVU	*LL_IWDG_IsActiveFlag_RVU*
		LL_IWDG_IsReady

66.12 PWR

Table 36: Correspondence between PWR registers and PWR low-layer driver functions

Register	Field	Function
CR	CSBF	*LL_PWR_ClearFlag_SB*
	CWUF	*LL_PWR_ClearFlag_WU*
	DBP	*LL_PWR_DisableBkUpAccess*
		LL_PWR_EnableBkUpAccess
		LL_PWR_IsEnabledBkUpAccess
	LPDS	*LL_PWR_GetPowerMode*
		LL_PWR_GetRegulModeDS
		LL_PWR_SetPowerMode
		LL_PWR_SetRegulModeDS
	PDDS	*LL_PWR_GetPowerMode*
		LL_PWR_SetPowerMode
	PLS	*LL_PWR_GetPVDLevel*
		LL_PWR_SetPVDLevel
	PVDE	*LL_PWR_DisablePVD*
		LL_PWR_EnablePVD
		LL_PWR_IsEnabledPVD
CSR	EWUP	*LL_PWR_DisableWakeUpPin*
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	PVDO	*LL_PWR_IsActiveFlag_PVDO*
	SBF	*LL_PWR_IsActiveFlag_SB*
	WUF	*LL_PWR_IsActiveFlag_WU*

66.13 RCC

Table 37: Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
BDCR	BDRST	*LL_RCC_ForceBackupDomainReset*
		LL_RCC_ReleaseBackupDomainReset
	LSEBYP	*LL_RCC_LSE_DisableBypass*
		LL_RCC_LSE_EnableBypass
	LSEON	*LL_RCC_LSE_Disable*
		LL_RCC_LSE_Enable
	LSERDY	*LL_RCC_LSE_IsReady*
	RTCEN	*LL_RCC_DisableRTC*

Register	Field	Function
		LL_RCC_EnableRTC
		LL_RCC_IsEnabledRTC
	RTCSEL	*LL_RCC_GetRTCClockSource*
		LL_RCC_SetRTCClockSource
CFGR	ADCPRE	*LL_RCC_GetADCClockSource*
		LL_RCC_SetADCClockSource
	HPRE	*LL_RCC_GetAHBPrescaler*
		LL_RCC_SetAHBPrescaler
	MCO	*LL_RCC_ConfigMCO*
	OTGFSPRE	*LL_RCC_GetUSBClockSource*
		LL_RCC_SetUSBClockSource
	PLLMULL	*LL_RCC_PLL_ConfigDomain_SYS*
		LL_RCC_PLL_GetMultiplicator
	PLLSRC	*LL_RCC_PLL_ConfigDomain_SYS*
		LL_RCC_PLL_GetMainSource
	PLLXTPRE	*LL_RCC_PLL_ConfigDomain_SYS*
	PPRE1	*LL_RCC_GetAPB1Prescaler*
		LL_RCC_SetAPB1Prescaler
	PPRE2	*LL_RCC_GetAPB2Prescaler*
		LL_RCC_SetAPB2Prescaler
	SW	*LL_RCC_SetSysClkSource*
	SWS	*LL_RCC_GetSysClkSource*
	USBPRE	*LL_RCC_GetUSBClockSource*
		LL_RCC_SetUSBClockSource
CFGR2	PLLXTPRE	*LL_RCC_PLL_GetPrediv*
	PREDIV1	*LL_RCC_PLL_ConfigDomain_SYS*
		LL_RCC_PLL_GetPrediv
	PREDIV1SRC	*LL_RCC_PLL_ConfigDomain_SYS*
		LL_RCC_PLL_GetMainSource
CIR	CSSC	*LL_RCC_ClearFlag_HSECSS*
	CSSF	*LL_RCC_IsActiveFlag_HSECSS*
	HSERDYC	*LL_RCC_ClearFlag_HSERDY*
	HSERDYF	*LL_RCC_IsActiveFlag_HSERDY*
	HSERDYIE	*LL_RCC_DisableIT_HSERDY*
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY

Register	Field	Function
	HSIRDYC	*LL_RCC_ClearFlag_HSIRDY*
	HSIRDYF	*LL_RCC_IsActiveFlag_HSIRDY*
	HSIRDYIE	*LL_RCC_DisableIT_HSIRDY*
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSERDYC	*LL_RCC_ClearFlag_LSERDY*
	LSERDYF	*LL_RCC_IsActiveFlag_LSERDY*
	LSERDYIE	*LL_RCC_DisableIT_LSERDY*
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYC	*LL_RCC_ClearFlag_LSIRDY*
	LSIRDYF	*LL_RCC_IsActiveFlag_LSIRDY*
	LSIRDYIE	*LL_RCC_DisableIT_LSIRDY*
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
	PLLRDYC	*LL_RCC_ClearFlag_PLLRDY*
	PLLRDYF	*LL_RCC_IsActiveFlag_PLLRDY*
	PLLRDYIE	*LL_RCC_DisableIT_PLLRDY*
		LL_RCC_EnableIT_PLLRDY
		LL_RCC_IsEnabledIT_PLLRDY
CR	CSSON	*LL_RCC_HSE_EnableCSS*
	HSEBYP	*LL_RCC_HSE_DisableBypass*
		LL_RCC_HSE_EnableBypass
	HSEON	*LL_RCC_HSE_Disable*
		LL_RCC_HSE_Enable
	HSERDY	*LL_RCC_HSE_IsReady*
	HSICAL	*LL_RCC_HSI_GetCalibration*
	HSION	*LL_RCC_HSI_Disable*
		LL_RCC_HSI_Enable
	HSIRDY	*LL_RCC_HSI_IsReady*
	HSITRIM	*LL_RCC_HSI_GetCalibTrimming*
		LL_RCC_HSI_SetCalibTrimming
	PLLON	*LL_RCC_PLL_Disable*
		LL_RCC_PLL_Enable
	PLLRDY	*LL_RCC_PLL_IsReady*
CSR	IWDGRSTF	*LL_RCC_IsActiveFlag_IWDGRST*

Register	Field	Function
	LPWRRSTF	*LL_RCC_IsActiveFlag_LPWRRST*
	LSION	*LL_RCC_LSI_Disable*
		LL_RCC_LSI_Enable
	LSIRDY	*LL_RCC_LSI_IsReady*
	PINRSTF	*LL_RCC_IsActiveFlag_PINRST*
	PORRSTF	*LL_RCC_IsActiveFlag_PORRST*
	RMVF	*LL_RCC_ClearResetFlags*
	SFTRSTF	*LL_RCC_IsActiveFlag_SFTRST*
	WWDGRSTF	*LL_RCC_IsActiveFlag_WWDGRST*

66.14 RTC

Table 38: Correspondence between RTC registers and RTC low-layer driver functions

Register	Field	Function
ALRH	ALR	*LL_RTC_ALARM_Get*
		LL_RTC_ALARM_Set
ALRL	ALR	*LL_RTC_ALARM_Get*
		LL_RTC_ALARM_Set
BKPDR	DR	*LL_RTC_BKP_GetRegister*
		LL_RTC_BKP_SetRegister
CNTH	CNT	*LL_RTC_TIME_Get*
		LL_RTC_TIME_Set
CNTL	CNT	*LL_RTC_TIME_Get*
		LL_RTC_TIME_Set
CR	TPAL	*LL_RTC_TAMPER_SetActiveLevel*
	TPE	*LL_RTC_TAMPER_Disable*
		LL_RTC_TAMPER_Enable
CRH	ALRIE	*LL_RTC_DisableIT_ALR*
		LL_RTC_EnableIT_ALR
		LL_RTC_IsEnabledIT_ALR
	OWIE	*LL_RTC_DisableIT_OW*
		LL_RTC_EnableIT_OW
		LL_RTC_IsEnabledIT_OW
	SECIE	*LL_RTC_DisableIT_SEC*
		LL_RTC_EnableIT_SEC
		LL_RTC_IsEnabledIT_SEC
CRL	ALRF	*LL_RTC_ClearFlag_ALR*

Register	Field	Function
		LL_RTC_IsActiveFlag_ALR
	CNF	*LL_RTC_EnableWriteProtection*
	OWF	*LL_RTC_ClearFlag_OW*
		LL_RTC_IsActiveFlag_OW
	RSF	*LL_RTC_ClearFlag_RS*
		LL_RTC_IsActiveFlag_RS
	RTC_CRL_CNF	*LL_RTC_DisableWriteProtection*
	RTOFF	*LL_RTC_IsActiveFlag_RTOF*
	SECF	*LL_RTC_ClearFlag_SEC*
		LL_RTC_IsActiveFlag_SEC
CSR	CTE	*LL_RTC_ClearFlag_TAMPE*
	CTI	*LL_RTC_ClearFlag_TAMPI*
	TEF	*LL_RTC_IsActiveFlag_TAMPE*
	TIF	*LL_RTC_IsActiveFlag_TAMPI*
	TPIE	*LL_RTC_EnableIT_TAMP*
		LL_RTC_IsEnabledIT_TAMP
DIVH	DIV	*LL_RTC_GetDivider*
DIVL	DIV	*LL_RTC_GetDivider*
PRLH	PRL	*LL_RTC_SetAsynchPrescaler*
PRLL	PRL	*LL_RTC_SetAsynchPrescaler*
RTCCR	ASOE	*LL_RTC_GetOutPutSource*
		LL_RTC_SetOutputSource
	ASOS	*LL_RTC_GetOutPutSource*
		LL_RTC_SetOutputSource
	CAL	*LL_RTC_CAL_SetCoarseDigital*
	CCO	*LL_RTC_GetOutPutSource*
		LL_RTC_SetOutputSource

66.15 SPI

Table 39: Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CR1	BIDIMODE	*LL_SPI_GetTransferDirection*
		LL_SPI_SetTransferDirection
	BIDIOE	*LL_SPI_GetTransferDirection*
		LL_SPI_SetTransferDirection
	BR	*LL_SPI_GetBaudRatePrescaler*

Register	Field	Function
		LL_SPI_SetBaudRatePrescaler
	CPHA	*LL_SPI_GetClockPhase*
		LL_SPI_SetClockPhase
	CPOL	*LL_SPI_GetClockPolarity*
		LL_SPI_SetClockPolarity
	CRCEN	*LL_SPI_DisableCRC*
		LL_SPI_EnableCRC
		LL_SPI_IsEnabledCRC
	CRCNEXT	*LL_SPI_SetCRCNext*
	DFF	*LL_SPI_GetDataWidth*
		LL_SPI_SetDataWidth
	LSBFIRST	*LL_SPI_GetTransferBitOrder*
		LL_SPI_SetTransferBitOrder
	MSTR	*LL_SPI_GetMode*
		LL_SPI_SetMode
	RXONLY	*LL_SPI_GetTransferDirection*
		LL_SPI_SetTransferDirection
	SPE	*LL_SPI_Disable*
		LL_SPI_Enable
		LL_SPI_IsEnabled
	SSI	*LL_SPI_GetMode*
		LL_SPI_SetMode
	SSM	*LL_SPI_GetNSSMode*
		LL_SPI_SetNSSMode
CR2	ERRIE	*LL_SPI_DisableIT_ERR*
		LL_SPI_EnableIT_ERR
		LL_SPI_IsEnabledIT_ERR
	RXDMAEN	*LL_SPI_DisableDMAReq_RX*
		LL_SPI_EnableDMAReq_RX
		LL_SPI_IsEnabledDMAReq_RX
	RXNEIE	*LL_SPI_DisableIT_RXNE*
		LL_SPI_EnableIT_RXNE
		LL_SPI_IsEnabledIT_RXNE
	SSOE	*LL_SPI_GetNSSMode*
		LL_SPI_SetNSSMode
	TXDMAEN	*LL_SPI_DisableDMAReq_TX*

Register	Field	Function
		LL_SPI_EnableDMAReq_TX
		LL_SPI_IsEnabledDMAReq_TX
	TXEIE	*LL_SPI_DisableIT_TXE*
		LL_SPI_EnableIT_TXE
		LL_SPI_IsEnabledIT_TXE
CRCPR	CRCPOLY	*LL_SPI_GetCRCPolynomial*
		LL_SPI_SetCRCPolynomial
DR	DR	*LL_SPI_DMA_GetRegAddr*
		LL_SPI_ReceiveData16
		LL_SPI_ReceiveData8
		LL_SPI_TransmitData16
		LL_SPI_TransmitData8
RXCRCR	RXCRC	*LL_SPI_GetRxCRC*
SR	BSY	*LL_SPI_IsActiveFlag_BSY*
	CRCERR	*LL_SPI_ClearFlag_CRCERR*
		LL_SPI_IsActiveFlag_CRCERR
	MODF	*LL_SPI_ClearFlag_MODF*
		LL_SPI_IsActiveFlag_MODF
	OVR	*LL_SPI_ClearFlag_OVR*
		LL_SPI_IsActiveFlag_OVR
	RXNE	*LL_SPI_IsActiveFlag_RXNE*
	TXE	*LL_SPI_IsActiveFlag_TXE*
TXCRCR	TXCRC	*LL_SPI_GetTxCRC*

66.16 SYSTEM

Table 40: Correspondence between SYSTEM registers and SYSTEM low-layer driver functions

Register	Field	Function
DBGMCU_CR	DBG_SLEEP	*LL_DBGMCU_DisableDBGSleepMode*
		LL_DBGMCU_EnableDBGSleepMode
	DBG_STANDBY	*LL_DBGMCU_DisableDBGStandbyMode*
		LL_DBGMCU_EnableDBGStandbyMode
	DBG_STOP	*LL_DBGMCU_DisableDBGStopMode*
		LL_DBGMCU_EnableDBGStopMode
	TRACE_IOEN	*LL_DBGMCU_GetTracePinAssignment*
		LL_DBGMCU_SetTracePinAssignment
	TRACE_MODE	*LL_DBGMCU_GetTracePinAssignment*

Register	Field	Function
		LL_DBGMCU_SetTracePinAssignment
DBGMCU_CR_APB1	DBG_CAN1_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_CAN2_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_I2C1_SMBUS_TIMEOUT	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_I2C2_SMBUS_TIMEOUT	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_IWDG_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_RTC_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM12_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM13_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM14_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM2_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM3_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM4_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM5_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM6_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*

Register	Field	Function
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_TIM7_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
	DBG_WWDG_STOP	*LL_DBGMCU_APB1_GRP1_FreezePeriph*
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
DBGMCU_CR_APB2	DBG_TIM10_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM11_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM15_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM16_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM17_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM1_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM8_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
	DBG_TIM9_STOP	*LL_DBGMCU_APB2_GRP1_FreezePeriph*
DBGMCU_IDCODE	DEV_ID	*LL_DBGMCU_GetDeviceID*
	REV_ID	*LL_DBGMCU_GetRevisionID*
FLASH_ACR	HLFCYA	*LL_FLASH_DisableHalfCycleAccess*
		LL_FLASH_EnableHalfCycleAccess
		LL_FLASH_IsHalfCycleAccessEnabled
	LATENCY	*LL_FLASH_GetLatency*
		LL_FLASH_SetLatency
	PRFTBE	*LL_FLASH_DisablePrefetch*
		LL_FLASH_EnablePrefetch
	PRFTBS	*LL_FLASH_IsPrefetchEnabled*

66.17 TIM

Table 41: Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
ARR	ARR	*LL_TIM_GetAutoReload*
		LL_TIM_SetAutoReload
BDTR	AOE	*LL_TIM_DisableAutomaticOutput*
		LL_TIM_EnableAutomaticOutput
		LL_TIM_IsEnabledAutomaticOutput
	BKE	*LL_TIM_DisableBRK*
		LL_TIM_EnableBRK

Register	Field	Function
	BKP	*LL_TIM_ConfigBRK*
	DTG	*LL_TIM_OC_SetDeadTime*
	LOCK	*LL_TIM_CC_SetLockLevel*
	MOE	*LL_TIM_DisableAllOutputs*
		LL_TIM_EnableAllOutputs
		LL_TIM_IsEnabledAllOutputs
	OSSI	*LL_TIM_SetOffStates*
	OSSR	*LL_TIM_SetOffStates*
CCER	CC1E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NE	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NP	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC1P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NE	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NP	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity

Register	Field	Function
	CC2P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NE	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NP	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4E	*LL_TIM_CC_DisableChannel*
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC4P	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
CCMR1	CC1S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput

Register	Field	Function
	CC2S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC1F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC1PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC2F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC2PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC1CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC1FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC1M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC1PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC2CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC2FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC2M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC2PE	*LL_TIM_OC_DisablePreload*

Register	Field	Function
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR2	CC3S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC4S	*LL_TIM_IC_Config*
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC3F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC3PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC4F	*LL_TIM_IC_Config*
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC4PSC	*LL_TIM_IC_Config*
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC3CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC3FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC3M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC3PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC4CE	*LL_TIM_OC_DisableClear*
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear

Register	Field	Function
	OC4FE	*LL_TIM_OC_DisableFast*
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	*LL_TIM_OC_GetMode*
		LL_TIM_OC_SetMode
	OC4PE	*LL_TIM_OC_DisablePreload*
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCR1	CCR1	*LL_TIM_IC_GetCaptureCH1*
		LL_TIM_OC_GetCompareCH1
		LL_TIM_OC_SetCompareCH1
CCR2	CCR2	*LL_TIM_IC_GetCaptureCH2*
		LL_TIM_OC_GetCompareCH2
		LL_TIM_OC_SetCompareCH2
CCR3	CCR3	*LL_TIM_IC_GetCaptureCH3*
		LL_TIM_OC_GetCompareCH3
		LL_TIM_OC_SetCompareCH3
CCR4	CCR4	*LL_TIM_IC_GetCaptureCH4*
		LL_TIM_OC_GetCompareCH4
		LL_TIM_OC_SetCompareCH4
CNT	CNT	*LL_TIM_GetCounter*
		LL_TIM_SetCounter
CR1	ARPE	*LL_TIM_DisableARRPreload*
		LL_TIM_EnableARRPreload
		LL_TIM_IsEnabledARRPreload
	CEN	*LL_TIM_DisableCounter*
		LL_TIM_EnableCounter
		LL_TIM_IsEnabledCounter
	CKD	*LL_TIM_GetClockDivision*
		LL_TIM_SetClockDivision
	CMS	*LL_TIM_GetCounterMode*
		LL_TIM_SetCounterMode
	DIR	*LL_TIM_GetCounterMode*
		LL_TIM_GetDirection
		LL_TIM_SetCounterMode
	OPM	*LL_TIM_GetOnePulseMode*

Register	Field	Function
		LL_TIM_SetOnePulseMode
	UDIS	*LL_TIM_DisableUpdateEvent*
		LL_TIM_EnableUpdateEvent
		LL_TIM_IsEnabledUpdateEvent
	URS	*LL_TIM_GetUpdateSource*
		LL_TIM_SetUpdateSource
CR2	CCDS	*LL_TIM_CC_GetDMAReqTrigger*
		LL_TIM_CC_SetDMAReqTrigger
	CCPC	*LL_TIM_CC_DisablePreload*
		LL_TIM_CC_EnablePreload
	CCUS	*LL_TIM_CC_SetUpdate*
	MMS	*LL_TIM_SetTriggerOutput*
	OIS1	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS1N	*LL_TIM_OC_GetIdleState*
		LL_TIM_OC_SetIdleState
	OIS2	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2N	*LL_TIM_OC_GetIdleState*
		LL_TIM_OC_SetIdleState
	OIS3	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3N	*LL_TIM_OC_GetIdleState*
		LL_TIM_OC_SetIdleState
	OIS4	*LL_TIM_OC_ConfigOutput*
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	TI1S	*LL_TIM_IC_DisableXORCombination*
		LL_TIM_IC_EnableXORCombination
		LL_TIM_IC_IsEnabledXORCombination
DCR	DBA	*LL_TIM_ConfigDMABurst*
	DBL	*LL_TIM_ConfigDMABurst*
DIER	BIE	*LL_TIM_DisableIT_BRK*

Register	Field	Function
		LL_TIM_EnableIT_BRK
		LL_TIM_IsEnabledIT_BRK
	CC1DE	*LL_TIM_DisableDMAReq_CC1*
		LL_TIM_EnableDMAReq_CC1
		LL_TIM_IsEnabledDMAReq_CC1
	CC1IE	*LL_TIM_DisableIT_CC1*
		LL_TIM_EnableIT_CC1
		LL_TIM_IsEnabledIT_CC1
	CC2DE	*LL_TIM_DisableDMAReq_CC2*
		LL_TIM_EnableDMAReq_CC2
		LL_TIM_IsEnabledDMAReq_CC2
	CC2IE	*LL_TIM_DisableIT_CC2*
		LL_TIM_EnableIT_CC2
		LL_TIM_IsEnabledIT_CC2
	CC3DE	*LL_TIM_DisableDMAReq_CC3*
		LL_TIM_EnableDMAReq_CC3
		LL_TIM_IsEnabledDMAReq_CC3
	CC3IE	*LL_TIM_DisableIT_CC3*
		LL_TIM_EnableIT_CC3
		LL_TIM_IsEnabledIT_CC3
	CC4DE	*LL_TIM_DisableDMAReq_CC4*
		LL_TIM_EnableDMAReq_CC4
		LL_TIM_IsEnabledDMAReq_CC4
	CC4IE	*LL_TIM_DisableIT_CC4*
		LL_TIM_EnableIT_CC4
		LL_TIM_IsEnabledIT_CC4
	COMDE	*LL_TIM_DisableDMAReq_COM*
		LL_TIM_EnableDMAReq_COM
		LL_TIM_IsEnabledDMAReq_COM
	COMIE	*LL_TIM_DisableIT_COM*
		LL_TIM_EnableIT_COM
		LL_TIM_IsEnabledIT_COM
	TDE	*LL_TIM_DisableDMAReq_TRIG*
		LL_TIM_EnableDMAReq_TRIG
		LL_TIM_IsEnabledDMAReq_TRIG
	TIE	*LL_TIM_DisableIT_TRIG*

Register	Field	Function
		LL_TIM_EnableIT_TRIG
		LL_TIM_IsEnabledIT_TRIG
	UDE	*LL_TIM_DisableDMAReq_UPDATE*
		LL_TIM_EnableDMAReq_UPDATE
		LL_TIM_IsEnabledDMAReq_UPDATE
	UIE	*LL_TIM_DisableIT_UPDATE*
		LL_TIM_EnableIT_UPDATE
		LL_TIM_IsEnabledIT_UPDATE
EGR	BG	*LL_TIM_GenerateEvent_BRK*
	CC1G	*LL_TIM_GenerateEvent_CC1*
	CC2G	*LL_TIM_GenerateEvent_CC2*
	CC3G	*LL_TIM_GenerateEvent_CC3*
	CC4G	*LL_TIM_GenerateEvent_CC4*
	COMG	*LL_TIM_GenerateEvent_COM*
	TG	*LL_TIM_GenerateEvent_TRIG*
	UG	*LL_TIM_GenerateEvent_UPDATE*
PSC	PSC	*LL_TIM_GetPrescaler*
		LL_TIM_SetPrescaler
RCR	REP	*LL_TIM_GetRepetitionCounter*
		LL_TIM_SetRepetitionCounter
SMCR	ECE	*LL_TIM_DisableExternalClock*
		LL_TIM_EnableExternalClock
		LL_TIM_IsEnabledExternalClock
		LL_TIM_SetClockSource
	ETF	*LL_TIM_ConfigETR*
	ETP	*LL_TIM_ConfigETR*
	ETPS	*LL_TIM_ConfigETR*
	MSM	*LL_TIM_DisableMasterSlaveMode*
		LL_TIM_EnableMasterSlaveMode
		LL_TIM_IsEnabledMasterSlaveMode
	SMS	*LL_TIM_SetClockSource*
		LL_TIM_SetEncoderMode
		LL_TIM_SetSlaveMode
	TS	*LL_TIM_SetTriggerInput*
SR	BIF	*LL_TIM_ClearFlag_BRK*
		LL_TIM_IsActiveFlag_BRK

Register	Field	Function
	CC1IF	*LL_TIM_ClearFlag_CC1*
		LL_TIM_IsActiveFlag_CC1
	CC1OF	*LL_TIM_ClearFlag_CC1OVR*
		LL_TIM_IsActiveFlag_CC1OVR
	CC2IF	*LL_TIM_ClearFlag_CC2*
		LL_TIM_IsActiveFlag_CC2
	CC2OF	*LL_TIM_ClearFlag_CC2OVR*
		LL_TIM_IsActiveFlag_CC2OVR
	CC3IF	*LL_TIM_ClearFlag_CC3*
		LL_TIM_IsActiveFlag_CC3
	CC3OF	*LL_TIM_ClearFlag_CC3OVR*
		LL_TIM_IsActiveFlag_CC3OVR
	CC4IF	*LL_TIM_ClearFlag_CC4*
		LL_TIM_IsActiveFlag_CC4
	CC4OF	*LL_TIM_ClearFlag_CC4OVR*
		LL_TIM_IsActiveFlag_CC4OVR
	COMIF	*LL_TIM_ClearFlag_COM*
		LL_TIM_IsActiveFlag_COM
	TIF	*LL_TIM_ClearFlag_TRIG*
		LL_TIM_IsActiveFlag_TRIG
	UIF	*LL_TIM_ClearFlag_UPDATE*
		LL_TIM_IsActiveFlag_UPDATE

66.18 USART

Table 42: Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
BRR	BRR	*LL_USART_GetBaudRate*
		LL_USART_SetBaudRate
CR1	IDLEIE	*LL_USART_DisableIT_IDLE*
		LL_USART_EnableIT_IDLE
		LL_USART_IsEnabledIT_IDLE
	M	*LL_USART_ConfigCharacter*
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	PCE	*LL_USART_ConfigCharacter*
		LL_USART_GetParity

Register	Field	Function
		LL_USART_SetParity
	PEIE	*LL_USART_DisableIT_PE*
		LL_USART_EnableIT_PE
		LL_USART_IsEnabledIT_PE
	PS	*LL_USART_ConfigCharacter*
		LL_USART_GetParity
		LL_USART_SetParity
	RE	*LL_USART_DisableDirectionRx*
		LL_USART_EnableDirectionRx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	RWU	*LL_USART_IsActiveFlag_RWU*
		LL_USART_RequestEnterMuteMode
		LL_USART_RequestExitMuteMode
	RXNEIE	*LL_USART_DisableIT_RXNE*
		LL_USART_EnableIT_RXNE
		LL_USART_IsEnabledIT_RXNE
	SBK	*LL_USART_IsActiveFlag_SBK*
		LL_USART_RequestBreakSending
	TCIE	*LL_USART_DisableIT_TC*
		LL_USART_EnableIT_TC
		LL_USART_IsEnabledIT_TC
	TE	*LL_USART_DisableDirectionTx*
		LL_USART_EnableDirectionTx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	TXEIE	*LL_USART_DisableIT_TXE*
		LL_USART_EnableIT_TXE
		LL_USART_IsEnabledIT_TXE
	UE	*LL_USART_Disable*
		LL_USART_Enable
		LL_USART_IsEnabled
	WAKE	*LL_USART_GetWakeUpMethod*
		LL_USART_SetWakeUpMethod
CR2	ADD	*LL_USART_GetNodeAddress*
		LL_USART_SetNodeAddress

Register	Field	Function
	CLKEN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSCLKOutput
		LL_USART_EnableSCLKOutput
		LL_USART_IsEnabledSCLKOutput
	CPHA	*LL_USART_ConfigClock*
		LL_USART_GetClockPhase
		LL_USART_SetClockPhase
	CPOL	*LL_USART_ConfigClock*
		LL_USART_GetClockPolarity
		LL_USART_SetClockPolarity
	LBCL	*LL_USART_ConfigClock*
		LL_USART_GetLastClkPulseOutput
		LL_USART_SetLastClkPulseOutput
	LBDIE	*LL_USART_DisableIT_LBD*
		LL_USART_EnableIT_LBD
		LL_USART_IsEnabledIT_LBD
	LBDL	*LL_USART_GetLINBrkDetectionLen*
		LL_USART_SetLINBrkDetectionLen
	LINEN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableLIN
		LL_USART_EnableLIN
		LL_USART_IsEnabledLIN
	STOP	*LL_USART_ConfigCharacter*
		LL_USART_ConfigIrdaMode

Register	Field	Function
		LL_USART_ConfigLINMode
		LL_USART_ConfigSmartcardMode
		LL_USART_GetStopBitsLength
		LL_USART_SetStopBitsLength
CR3	CTSE	*LL_USART_DisableCTSHWFlowCtrl*
		LL_USART_EnableCTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	CTSIE	*LL_USART_DisableIT_CTS*
		LL_USART_EnableIT_CTS
		LL_USART_IsEnabledIT_CTS
	DMAR	*LL_USART_DisableDMAReq_RX*
		LL_USART_EnableDMAReq_RX
		LL_USART_IsEnabledDMAReq_RX
	DMAT	*LL_USART_DisableDMAReq_TX*
		LL_USART_EnableDMAReq_TX
		LL_USART_IsEnabledDMAReq_TX
	EIE	*LL_USART_DisableIT_ERROR*
		LL_USART_EnableIT_ERROR
		LL_USART_IsEnabledIT_ERROR
	HDSEL	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableHalfDuplex
		LL_USART_EnableHalfDuplex
		LL_USART_IsEnabledHalfDuplex
	IREN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSyncMode

Register	Field	Function
		LL_USART_DisableIrda
		LL_USART_EnableIrda
		LL_USART_IsEnabledIrda
	IRLP	*LL_USART_GetIrdaPowerMode*
		LL_USART_SetIrdaPowerMode
	NACK	*LL_USART_DisableSmartcardNACK*
		LL_USART_EnableSmartcardNACK
		LL_USART_IsEnabledSmartcardNACK
	RTSE	*LL_USART_DisableRTSHWFlowCtrl*
		LL_USART_EnableRTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	SCEN	*LL_USART_ConfigAsyncMode*
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSmartcard
		LL_USART_EnableSmartcard
		LL_USART_IsEnabledSmartcard
DR	DR	*LL_USART_DMA_GetRegAddr*
		LL_USART_ReceiveData8
		LL_USART_ReceiveData9
		LL_USART_TransmitData8
		LL_USART_TransmitData9
GTPR	GT	*LL_USART_GetSmartcardGuardTime*
		LL_USART_SetSmartcardGuardTime
	PSC	*LL_USART_GetIrdaPrescaler*
		LL_USART_GetSmartcardPrescaler
		LL_USART_SetIrdaPrescaler
		LL_USART_SetSmartcardPrescaler
SR	CTS	*LL_USART_ClearFlag_nCTS*
		LL_USART_IsActiveFlag_nCTS
	FE	*LL_USART_ClearFlag_FE*

Register	Field	Function
		LL_USART_IsActiveFlag_FE
	IDLE	*LL_USART_ClearFlag_IDLE*
		LL_USART_IsActiveFlag_IDLE
	LBD	*LL_USART_ClearFlag_LBD*
		LL_USART_IsActiveFlag_LBD
	NF	*LL_USART_ClearFlag_NE*
		LL_USART_IsActiveFlag_NE
	ORE	*LL_USART_ClearFlag_ORE*
		LL_USART_IsActiveFlag_ORE
	PE	*LL_USART_ClearFlag_PE*
		LL_USART_IsActiveFlag_PE
	RXNE	*LL_USART_ClearFlag_RXNE*
		LL_USART_IsActiveFlag_RXNE
	TC	*LL_USART_ClearFlag_TC*
		LL_USART_IsActiveFlag_TC
	TXE	*LL_USART_IsActiveFlag_TXE*

66.19 WWDG

Table 43: Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	*LL_WWDG_EnableIT_EWKUP*
		LL_WWDG_IsEnabledIT_EWKUP
	W	*LL_WWDG_GetWindow*
		LL_WWDG_SetWindow
	WDGTB	*LL_WWDG_GetPrescaler*
		LL_WWDG_SetPrescaler
CR	T	*LL_WWDG_GetCounter*
		LL_WWDG_SetCounter
	WDGA	*LL_WWDG_Enable*
		LL_WWDG_IsEnabled
SR	EWIF	*LL_WWDG_ClearFlag_EWKUP*
		LL_WWDG_IsActiveFlag_EWKUP

67 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 STM32F1 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F1 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *Section 2.4: "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: stm32f1xx_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 (stm32f1xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32f1xx_hal.h file has to be included.

What is the difference between stm32f1xx_hal_ppp.c/.h and stm32f1xx_hal_ppp_*ex*.c/.h?

The HAL driver architecture supports common features across STM32 series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

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 inxx_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.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32f1xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32l4xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

68 Revision history

Table 44: Document revision history

Date	Revision	Changes
06-Jan-2015	1	Initial release.
20-Apr-2017	2	Updated *Table 5: "List of devices supported by HAL drivers"* to add new supported part numbers and LL drivers Added description of LL Generic drivers. Corrected typo in *Section 2.11.6: "DMA"*.

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.

© 2017 STMicroelectronics – All rights reserved