

## Description of STM32L0 HAL and Low-layer drivers

### Introduction

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

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL0 for STM32L0 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software ensuring maximized portability across STM32 portfolio. The HAL is available for all peripherals.
  - The Low Layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB.
  - All embedded software utilities coming with a full set of examples.

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

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 USB).

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

<b>1</b>	<b>Acronyms and definitions.....</b>	<b>25</b>
<b>2</b>	<b>Overview of HAL drivers .....</b>	<b>27</b>
2.1	HAL and user-application files.....	27
2.1.1	HAL driver files .....	27
2.1.2	User-application files .....	28
2.2	HAL data structures .....	30
2.2.1	Peripheral handle structures .....	30
2.2.2	Initialization and configuration structure .....	32
2.2.3	Specific process structures .....	32
2.3	API classification .....	32
2.4	Devices supported by HAL drivers .....	33
2.5	HAL drivers 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 .....	43
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.....	51
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
<b>3</b>	<b>Overview of Low Layer drivers.....</b>	<b>63</b>

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 .....	68
<b>4</b>	<b>HAL and LL cohabitation .....</b>	<b>71</b>
4.1	Low Layer driver used in standalone mode.....	71
4.2	Mixed use of Low Layer APIs and HAL drivers .....	71
<b>5</b>	<b>HAL System Driver .....</b>	<b>72</b>
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.....	78
5.2.1	HAL.....	78
<b>6</b>	<b>HAL ADC Generic Driver.....</b>	<b>82</b>
6.1	ADC Firmware driver registers structures .....	82
6.1.1	ADC_OversamplingTypeDef .....	82
6.1.2	ADC_InitTypeDef.....	82
6.1.3	ADC_HandleTypeDef .....	84
6.1.4	ADC_ChannelConfTypeDef .....	85
6.1.5	ADC_AnalogWDGConfTypeDef.....	85
6.2	ADC Firmware driver API description.....	86
6.2.1	ADC peripheral features .....	86
6.2.2	How to use this driver .....	86
6.2.3	IO operation functions .....	89
6.2.4	Peripheral Control functions .....	89
6.2.5	ADC Peripheral State functions.....	89
6.2.6	Detailed description of functions .....	90
6.3	ADC Firmware driver defines .....	95
6.3.1	ADC .....	95
<b>7</b>	<b>HAL ADC Extension Driver .....</b>	<b>108</b>
7.1	ADCEx Firmware driver API description .....	108
7.1.1	ADC specific features .....	108
7.1.2	How to use this driver .....	108
7.1.3	Detailed description of functions .....	108

7.2	ADCEx Firmware driver defines .....	110
7.2.1	ADCEx .....	110
<b>8</b>	<b>HAL COMP Generic Driver.....</b>	<b>111</b>
8.1	COMP Firmware driver registers structures .....	111
8.1.1	COMP_InitTypeDef .....	111
8.1.2	COMP_HandleTypeDef .....	111
8.2	COMP Firmware driver API description .....	112
8.2.1	COMP Peripheral features .....	112
8.2.2	How to use this driver .....	112
8.2.3	Initialization and de-initialization functions .....	113
8.2.4	IO operation functions .....	113
8.2.5	Peripheral Control functions .....	113
8.2.6	Peripheral State functions .....	113
8.2.7	Detailed description of functions .....	114
8.3	COMP Firmware driver defines .....	116
8.3.1	COMP .....	116
<b>9</b>	<b>HAL COMP Extension Driver .....</b>	<b>124</b>
9.1	COMPEX Firmware driver API description .....	124
9.1.1	COMP peripheral Extended features .....	124
9.1.2	Detailed description of functions .....	124
<b>10</b>	<b>HAL CORTEX Generic Driver.....</b>	<b>125</b>
10.1	CORTEX Firmware driver registers structures .....	125
10.1.1	MPU_Region_InitTypeDef .....	125
10.2	CORTEX Firmware driver API description .....	126
10.2.1	How to use this driver .....	126
10.2.2	Initialization and de-initialization functions .....	126
10.2.3	Peripheral Control functions .....	127
10.2.4	Detailed description of functions .....	127
10.3	CORTEX Firmware driver defines .....	130
10.3.1	CORTEX .....	130
<b>11</b>	<b>HAL CRC Generic Driver.....</b>	<b>133</b>
11.1	CRC Firmware driver registers structures .....	133
11.1.1	CRC_InitTypeDef .....	133
11.1.2	CRC_HandleTypeDef .....	134
11.2	CRC Firmware driver API description .....	134
11.2.1	Initialization and de-initialization functions .....	134
11.2.2	Peripheral Control functions .....	135

11.2.3	Peripheral State functions .....	135
11.2.4	Detailed description of functions .....	135
11.3	CRC Firmware driver defines .....	137
11.3.1	CRC .....	137
<b>12</b>	<b>HAL CRC Extension Driver .....</b>	<b>139</b>
12.1	CRCEx Firmware driver API description .....	139
12.1.1	CRC Extended features functions .....	139
12.1.2	Detailed description of functions .....	139
12.2	CRCEx Firmware driver defines .....	140
12.2.1	CRCEx .....	140
<b>13</b>	<b>HAL CRYP Generic Driver.....</b>	<b>142</b>
13.1	CRYP Firmware driver registers structures .....	142
13.1.1	CRYP_InitTypeDef .....	142
13.1.2	CRYP_HandleTypeDef .....	142
13.2	CRYP Firmware driver API description .....	143
13.2.1	Initialization and de-initialization functions .....	143
13.2.2	AES processing functions .....	143
13.2.3	CRYP IRQ handler management .....	144
13.2.4	Peripheral State functions .....	144
13.2.5	DMA callback functions .....	144
13.2.6	Detailed description of functions .....	144
13.3	CRYP Firmware driver defines .....	152
13.3.1	CRYP .....	152
<b>14</b>	<b>HAL CRYP Extension Driver .....</b>	<b>156</b>
14.1	CRYPEx Firmware driver API description .....	156
14.1.1	Extended features functions .....	156
14.1.2	Detailed description of functions .....	156
<b>15</b>	<b>HAL DAC Generic Driver.....</b>	<b>157</b>
15.1	DAC Firmware driver registers structures .....	157
15.1.1	DAC_HandleTypeDef .....	157
15.1.2	DAC_ChannelConfTypeDef .....	157
15.2	DAC Firmware driver API description.....	158
15.2.1	DAC Peripheral features .....	158
15.2.2	How to use this driver .....	159
15.2.3	Initialization and de-initialization functions .....	160
15.2.4	IO operation functions .....	160
15.2.5	Peripheral Control functions .....	161

15.2.6	Peripheral State and Errors functions .....	161
15.2.7	Detailed description of functions .....	161
15.3	DAC Firmware driver defines .....	166
15.3.1	DAC .....	166
<b>16</b>	<b>HAL DAC Extension Driver .....</b>	<b>169</b>
16.1	DACEx Firmware driver API description .....	169
16.1.1	How to use this driver .....	169
16.1.2	Detailed description of functions .....	169
16.2	DACEx Firmware driver defines .....	172
16.2.1	DACEx .....	172
<b>17</b>	<b>HAL DMA Generic Driver .....</b>	<b>174</b>
17.1	DMA Firmware driver registers structures .....	174
17.1.1	DMA_InitTypeDef .....	174
17.1.2	__DMA_HandleTypeDef .....	175
17.2	DMA Firmware driver API description .....	175
17.2.1	Initialization and de-initialization functions .....	175
17.2.2	IO operation functions .....	176
17.2.3	Peripheral State functions .....	176
17.2.4	Detailed description of functions .....	176
17.3	DMA Firmware driver defines .....	178
17.3.1	DMA .....	178
<b>18</b>	<b>HAL FIREWALL Generic Driver .....</b>	<b>185</b>
18.1	FIREWALL Firmware driver registers structures .....	185
18.1.1	FIREWALL_InitTypeDef .....	185
18.2	FIREWALL Firmware driver API description .....	186
18.2.1	How to use this driver .....	186
18.2.2	Initialization and Configuration functions .....	186
18.2.3	Detailed description of functions .....	186
18.3	FIREWALL Firmware driver defines .....	188
18.3.1	FIREWALL .....	188
<b>19</b>	<b>HAL FLASH Generic Driver .....</b>	<b>193</b>
19.1	FLASH Firmware driver registers structures .....	193
19.1.1	FLASH_EraseInitTypeDef .....	193
19.1.2	FLASH_ProcessTypeDef .....	193
19.2	FLASH Firmware driver API description .....	194
19.2.1	FLASH peripheral features .....	194

19.2.2	How to use this driver .....	194
19.2.3	Programming operation functions .....	195
19.2.4	Option Bytes Programming functions.....	195
19.2.5	Peripheral Control functions .....	196
19.2.6	Peripheral Errors functions .....	196
19.2.7	Detailed description of functions .....	196
19.3	FLASH Firmware driver defines .....	199
19.3.1	FLASH .....	199
<b>20</b>	<b>HAL FLASH Extension Driver .....</b>	<b>203</b>
20.1	FLASHEx Firmware driver registers structures .....	203
20.1.1	FLASH_OBProgramInitTypeDef .....	203
20.1.2	FLASH_AdvOBProgramInitTypeDef .....	204
20.2	FLASHEx Firmware driver API description.....	204
20.2.1	Flash peripheral Extended features .....	204
20.2.2	How to use this driver .....	204
20.2.3	FLASH Erasing Programming functions.....	205
20.2.4	Option Bytes Programming functions.....	205
20.2.5	DATA EEPROM Programming functions .....	206
20.2.6	Detailed description of functions .....	206
20.3	FLASHEx Firmware driver defines .....	209
20.3.1	FLASHEx .....	209
<b>21</b>	<b>HAL FLASH__RAMFUNC Generic Driver.....</b>	<b>216</b>
21.1	FLASH__RAMFUNC Firmware driver API description.....	216
21.1.1	Detailed description of functions .....	216
<b>22</b>	<b>HAL GPIO Generic Driver.....</b>	<b>219</b>
22.1	GPIO Firmware driver registers structures .....	219
22.1.1	GPIO_InitTypeDef .....	219
22.2	GPIO Firmware driver API description .....	219
22.2.1	GPIO Peripheral features .....	219
22.2.2	How to use this driver .....	220
22.2.3	Initialization and de-initialization functions .....	220
22.2.4	IO operation functions .....	221
22.2.5	Detailed description of functions .....	221
22.3	GPIO Firmware driver defines.....	223
22.3.1	GPIO .....	223
<b>23</b>	<b>HAL GPIO Extension Driver .....</b>	<b>227</b>
23.1	GPIOEx Firmware driver defines.....	227

23.1.1	GPIOEx .....	227
<b>24 HAL I2C Generic Driver .....</b>	<b>229</b>	
24.1	I2C Firmware driver registers structures .....	229
24.1.1	I2C_InitTypeDef.....	229
24.1.2	I2C_HandleTypeDef .....	229
24.2	I2C Firmware driver API description.....	230
24.2.1	How to use this driver .....	230
24.2.2	Initialization and de-initialization functions .....	233
24.2.3	IO operation functions .....	233
24.2.4	Peripheral State and Errors functions .....	235
24.2.5	Detailed description of functions .....	235
24.3	I2C Firmware driver defines .....	243
24.3.1	I2C .....	243
<b>25 HAL I2C Extension Driver .....</b>	<b>249</b>	
25.1	I2CEx Firmware driver API description .....	249
25.1.1	I2C peripheral Extended features.....	249
25.1.2	How to use this driver .....	249
25.1.3	Extended features functions .....	249
25.1.4	Detailed description of functions .....	249
25.2	I2CEx Firmware driver defines .....	251
25.2.1	I2CEx .....	251
<b>26 HAL I2S Generic Driver .....</b>	<b>252</b>	
26.1	I2S Firmware driver registers structures .....	252
26.1.1	I2S_InitTypeDef.....	252
26.1.2	I2S_HandleTypeDef .....	252
26.2	I2S Firmware driver API description.....	253
26.2.1	How to use this driver .....	253
26.2.2	Initialization and de-initialization functions .....	255
26.2.3	IO operation functions .....	255
26.2.4	Peripheral State and Errors functions .....	256
26.2.5	Detailed description of functions .....	256
26.3	I2S Firmware driver defines .....	262
26.3.1	I2S .....	262
<b>27 HAL IRDA Generic Driver .....</b>	<b>267</b>	
27.1	IRDA Firmware driver registers structures .....	267
27.1.1	IRDA_InitTypeDef.....	267
27.1.2	IRDA_HandleTypeDef .....	267

27.2	IRDA Firmware driver API description.....	268
27.2.1	Initialization and Configuration functions .....	268
27.2.2	IO operation functions .....	269
27.2.3	Peripheral Control functions .....	269
27.2.4	Detailed description of functions .....	270
27.3	IRDA Firmware driver defines .....	274
27.3.1	IRDA .....	274
<b>28</b>	<b>HAL IRDA Extension Driver .....</b>	<b>284</b>
28.1	IRDAEx Firmware driver defines .....	284
28.1.1	IRDAEx .....	284
<b>29</b>	<b>HAL IWDG Generic Driver.....</b>	<b>285</b>
29.1	IWDG Firmware driver registers structures .....	285
29.1.1	IWDG_InitTypeDef .....	285
29.1.2	IWDG_HandleTypeDef.....	285
29.2	IWDG Firmware driver API description .....	286
29.2.1	IWDG Generic features .....	286
29.2.2	How to use this driver .....	286
29.2.3	Initialization and de-initialization functions .....	287
29.2.4	IO operation functions .....	287
29.2.5	Peripheral State functions .....	287
29.2.6	Detailed description of functions .....	287
29.3	IWDG Firmware driver defines .....	288
29.3.1	IWDG .....	288
<b>30</b>	<b>HAL LCD Generic Driver .....</b>	<b>291</b>
30.1	LCD Firmware driver registers structures.....	291
30.1.1	LCD_InitTypeDef .....	291
30.1.2	LCD_HandleTypeDef .....	292
30.2	LCD Firmware driver API description .....	292
30.2.1	How to use this driver .....	292
30.2.2	Initialization and Configuration functions .....	293
30.2.3	IO operation functions .....	293
30.2.4	Peripheral State functions .....	293
30.2.5	Detailed description of functions .....	294
30.3	LCD Firmware driver defines.....	296
30.3.1	LCD .....	296
<b>31</b>	<b>HAL LPTIM Generic Driver.....</b>	<b>306</b>
31.1	LPTIM Firmware driver registers structures .....	306

31.1.1	LPTIM_ClockConfigTypeDef .....	306
31.1.2	LPTIM_ULPClockConfigTypeDef .....	306
31.1.3	LPTIM_TriggerConfigTypeDef .....	306
31.1.4	LPTIM_InitTypeDef .....	307
31.1.5	LPTIM_HandleTypeDef .....	307
31.2	LPTIM Firmware driver API description .....	308
31.2.1	Initialization and de-initialization functions .....	308
31.2.2	LPTIM Start Stop operation functions .....	308
31.2.3	LPTIM Read operation functions .....	309
31.2.4	LPTIM IRQ handler .....	309
31.2.5	Peripheral State functions .....	309
31.2.6	Detailed description of functions .....	310
31.3	LPTIM Firmware driver defines .....	318
31.3.1	LPTIM .....	318
<b>32</b>	<b>HAL PCD Generic Driver .....</b>	<b>327</b>
32.1	PCD Firmware driver registers structures .....	327
32.1.1	PCD_InitTypeDef .....	327
32.1.2	PCD_EPTypeDef .....	327
32.1.3	PCD_HandleTypeDef .....	328
32.2	PCD Firmware driver API description .....	329
32.2.1	How to use this driver .....	329
32.2.2	Initialization and de-initialization functions .....	330
32.2.3	IO operation functions .....	330
32.2.4	Peripheral Control functions .....	330
32.2.5	Peripheral State functions .....	330
32.2.6	Detailed description of functions .....	331
32.3	PCD Firmware driver defines .....	337
32.3.1	PCD .....	337
<b>33</b>	<b>HAL PCD Extension Driver .....</b>	<b>346</b>
33.1	PCDEx Firmware driver API description .....	346
33.1.1	Peripheral extended features functions .....	346
33.1.2	Detailed description of functions .....	346
<b>34</b>	<b>HAL PWR Generic Driver .....</b>	<b>347</b>
34.1	PWR Firmware driver registers structures .....	347
34.1.1	PWR_PVTypeDef .....	347
34.2	PWR Firmware driver API description .....	347
34.2.1	Initialization and de-initialization functions .....	347

34.2.2	Peripheral Control functions .....	347
34.2.3	Detailed description of functions .....	351
34.3	PWR Firmware driver defines .....	355
34.3.1	PWR .....	355
<b>35</b>	<b>HAL PWR Extension Driver .....</b>	<b>361</b>
35.1	PWREx Firmware driver defines .....	361
35.1.1	PWREx .....	361
<b>36</b>	<b>HAL RCC Generic Driver.....</b>	<b>362</b>
36.1	RCC Firmware driver registers structures .....	362
36.1.1	RCC_PLLInitTypeDef .....	362
36.1.2	RCC_OscInitTypeDef .....	362
36.1.3	RCC_ClkInitTypeDef .....	363
36.2	RCC Firmware driver API description .....	364
36.2.1	RCC specific features .....	364
36.2.2	RCC Limitations.....	364
36.2.3	Initialization and de-initialization functions .....	364
36.2.4	Peripheral Control functions .....	365
36.2.5	Detailed description of functions .....	366
36.3	RCC Firmware driver defines .....	370
36.3.1	RCC .....	370
<b>37</b>	<b>HAL RCC Extension Driver .....</b>	<b>391</b>
37.1	RCCEx Firmware driver registers structures .....	391
37.1.1	RCC_PeriphCLKInitTypeDef .....	391
37.1.2	RCC_CRSInitTypeDef .....	392
37.1.3	RCC_CRSSynchroInfoTypeDef .....	392
37.2	RCCEx Firmware driver API description .....	393
37.2.1	Extended Peripheral Control functions .....	393
37.2.2	Detailed description of functions .....	393
37.3	RCCEx Firmware driver defines.....	396
37.3.1	RCCEx .....	396
<b>38</b>	<b>HAL RNG Generic Driver.....</b>	<b>419</b>
38.1	RNG Firmware driver registers structures .....	419
38.1.1	RNG_HandleTypeDef.....	419
38.2	RNG Firmware driver API description .....	419
38.2.1	How to use this driver .....	419
38.2.2	Initialization and de-initialization functions .....	419
38.2.3	Peripheral Control functions .....	420

38.2.4	Peripheral State functions .....	420
38.2.5	Detailed description of functions .....	420
38.3	RNG Firmware driver defines.....	423
38.3.1	RNG.....	423
<b>39</b>	<b>HAL RTC Generic Driver .....</b>	<b>426</b>
39.1	RTC Firmware driver registers structures .....	426
39.1.1	RTC_InitTypeDef.....	426
39.1.2	RTC_TimeTypeDef.....	426
39.1.3	RTC_DateTypeDef .....	427
39.1.4	RTC_AlarmTypeDef .....	428
39.1.5	RTC_HandleTypeDef .....	429
39.2	RTC Firmware driver API description.....	429
39.2.1	Backup Domain Operating Condition.....	429
39.2.2	Backup Domain Reset.....	429
39.2.3	Backup Domain Access.....	429
39.2.4	How to use RTC Driver.....	430
39.2.5	RTC and low power modes .....	430
39.2.6	Initialization and de-initialization functions .....	430
39.2.7	RTC Time and Date functions .....	431
39.2.8	RTC Alarm functions .....	431
39.2.9	Peripheral Control functions .....	431
39.2.10	Peripheral State functions .....	431
39.2.11	Detailed description of functions .....	432
39.3	RTC Firmware driver defines .....	437
39.3.1	RTC .....	437
<b>40</b>	<b>HAL RTC Extension Driver .....</b>	<b>447</b>
40.1	RTCEEx Firmware driver registers structures .....	447
40.1.1	RTC_TamperTypeDef .....	447
40.2	RTCEEx Firmware driver API description.....	448
40.2.1	RTC TimeStamp and Tamper functions.....	448
40.2.2	RTC Wake-up functions .....	448
40.2.3	Extended Peripheral Control functions.....	448
40.2.4	Extended features functions .....	449
40.2.5	Detailed description of functions .....	449
40.3	RTCEEx Firmware driver defines .....	458
40.3.1	RTCEEx .....	458
<b>41</b>	<b>HAL SMARTCARD Generic Driver.....</b>	<b>475</b>

41.1	SMARTCARD Firmware driver registers structures .....	475
41.1.1	SMARTCARD_InitTypeDef .....	475
41.1.2	SMARTCARD_AdvFeatureInitTypeDef.....	476
41.1.3	SMARTCARD_HandleTypeDef.....	477
41.2	SMARTCARD Firmware driver API description.....	478
41.2.1	Initialization and Configuration functions.....	478
41.2.2	IO operation functions .....	478
41.2.3	Peripheral State functions .....	479
41.2.4	Detailed description of functions .....	479
41.3	SMARTCARD Firmware driver defines .....	482
41.3.1	SMARTCARD.....	482
<b>42</b>	<b>HAL SMARTCARD Extension Driver .....</b>	<b>495</b>
42.1	SMARTCARDEX Firmware driver API description .....	495
42.1.1	How to use this driver.....	495
42.1.2	Peripheral Control functions .....	495
42.1.3	Detailed description of functions .....	495
42.2	SMARTCARDEX Firmware driver defines .....	496
42.2.1	SMARTCARDEX.....	496
<b>43</b>	<b>HAL SMBUS Generic Driver.....</b>	<b>497</b>
43.1	SMBUS Firmware driver registers structures .....	497
43.1.1	SMBUS_InitTypeDef .....	497
43.1.2	SMBUS_HandleTypeDef.....	498
43.2	SMBUS Firmware driver API description .....	499
43.2.1	Initialization and de-initialization functions .....	499
43.2.2	IO operation functions .....	499
43.2.3	Peripheral State and Errors functions .....	500
43.2.4	Detailed description of functions .....	500
43.3	SMBUS Firmware driver defines .....	506
43.3.1	SMBUS .....	506
<b>44</b>	<b>HAL SPI Generic Driver.....</b>	<b>514</b>
44.1	SPI Firmware driver registers structures .....	514
44.1.1	SPI_InitTypeDef .....	514
44.1.2	__SPI_HandleTypeDef.....	515
44.2	SPI Firmware driver API description .....	516
44.2.1	How to use this driver .....	516
44.2.2	Initialization and de-initialization functions .....	517
44.2.3	IO operation functions .....	517

44.2.4	Peripheral State and Errors functions .....	518
44.2.5	Detailed description of functions .....	518
44.3	SPI Firmware driver defines .....	524
44.3.1	SPI .....	524
<b>45</b>	<b>HAL TIM Generic Driver .....</b>	<b>530</b>
45.1	TIM Firmware driver registers structures.....	530
45.1.1	TIM_Base_InitTypeDef.....	530
45.1.2	TIM_OC_InitTypeDef.....	530
45.1.3	TIM_OnePulse_InitTypeDef .....	531
45.1.4	TIM_IC_InitTypeDef .....	531
45.1.5	TIM_Encoder_InitTypeDef .....	532
45.1.6	TIM_ClockConfigTypeDef .....	533
45.1.7	TIM_ClearInputConfigTypeDef.....	533
45.1.8	TIM_SlaveConfigTypeDef .....	534
45.1.9	TIM_HandleTypeDef .....	534
45.2	TIM Firmware driver API description .....	535
45.2.1	TIMER Generic features.....	535
45.2.2	How to use this driver .....	535
45.2.3	Timer Base functions.....	536
45.2.4	Peripheral State functions .....	536
45.2.5	Detailed description of functions .....	537
45.3	TIM Firmware driver defines.....	559
45.3.1	TIM.....	559
<b>46</b>	<b>HAL TIM Extension Driver.....</b>	<b>573</b>
46.1	TIMEx Firmware driver registers structures.....	573
46.1.1	TIM_MasterConfigTypeDef .....	573
46.2	TIMEx Firmware driver API description .....	573
46.2.1	TIM specific features integration .....	573
46.2.2	How to use this driver .....	573
46.2.3	Peripheral Control functions .....	574
46.2.4	Detailed description of functions .....	574
46.3	TIMEx Firmware driver defines .....	575
46.3.1	TIMEx .....	575
<b>47</b>	<b>HAL TSC Generic Driver .....</b>	<b>577</b>
47.1	TSC Firmware driver registers structures.....	577
47.1.1	TSC_InitTypeDef .....	577
47.1.2	TSC_IOConfigTypeDef.....	578

47.1.3	TSC_HandleTypeDef .....	578
47.2	TSC Firmware driver API description .....	578
47.2.1	TSC specific features .....	578
47.2.2	How to use this driver .....	579
47.2.3	IO Operation functions.....	579
47.2.4	Peripheral Control functions .....	580
47.2.5	State functions.....	580
47.2.6	Initialization and de-initialization functions .....	580
47.2.7	Detailed description of functions .....	580
47.3	TSC Firmware driver defines.....	584
47.3.1	TSC.....	584
<b>48</b>	<b>HAL UART Generic Driver.....</b>	<b>593</b>
48.1	UART Firmware driver registers structures .....	593
48.1.1	UART_InitTypeDef .....	593
48.1.2	UART_AdvFeatureInitTypeDef.....	594
48.1.3	UART_HandleTypeDef.....	594
48.2	UART Firmware driver API description .....	596
48.2.1	Initialization and Configuration functions.....	596
48.2.2	IO operation functions .....	596
48.2.3	Peripheral Control functions .....	597
48.2.4	Detailed description of functions .....	597
48.3	UART Firmware driver defines .....	604
48.3.1	UART .....	604
<b>49</b>	<b>HAL UART Extension Driver.....</b>	<b>622</b>
49.1	UARTEEx Firmware driver registers structures .....	622
49.1.1	UART_WakeUpTypeDef .....	622
49.2	UARTEEx Firmware driver API description .....	622
49.2.1	Initialization and Configuration functions.....	622
49.2.2	Peripheral Control funtions .....	622
49.2.3	Detailed description of functions .....	623
49.3	UARTEEx Firmware driver defines .....	625
49.3.1	UARTEEx .....	625
<b>50</b>	<b>HAL USART Generic Driver .....</b>	<b>627</b>
50.1	USART Firmware driver registers structures.....	627
50.1.1	USART_InitTypeDef .....	627
50.1.2	USART_HandleTypeDef .....	628
50.2	USART Firmware driver API description .....	629

50.2.1	Initialization and Configuration functions .....	629
50.2.2	IO operation functions .....	629
50.2.3	Peripheral State functions .....	630
50.2.4	Detailed description of functions .....	631
50.3	USART Firmware driver defines.....	636
50.3.1	USART.....	636
<b>51</b>	<b>HAL USART Extension Driver .....</b>	<b>646</b>
51.1	USARTEx Firmware driver defines .....	646
51.1.1	USARTEx .....	646
<b>52</b>	<b>HAL WWDG Generic Driver .....</b>	<b>647</b>
52.1	WWDG Firmware driver registers structures.....	647
52.1.1	WWDG_InitTypeDef .....	647
52.1.2	WWDG_HandleTypeDef .....	647
52.2	WWDG Firmware driver API description .....	648
52.2.1	WWDG specific features .....	648
52.2.2	How to use this driver .....	648
52.2.3	Initialization and de-initialization functions .....	648
52.2.4	IO operation functions .....	649
52.2.5	Peripheral State functions .....	649
52.2.6	Detailed description of functions .....	649
52.3	WWDG Firmware driver defines.....	651
52.3.1	WWDG.....	651
<b>53</b>	<b>LL ADC Generic Driver.....</b>	<b>656</b>
53.1	ADC Firmware driver registers structures .....	656
53.1.1	LL_ADC_CommonInitTypeDef .....	656
53.1.2	LL_ADC_InitTypeDef.....	656
53.1.3	LL_ADC_REG_InitTypeDef.....	657
53.2	ADC Firmware driver API description.....	658
53.2.1	Detailed description of functions .....	658
53.3	ADC Firmware driver defines .....	706
53.3.1	ADC .....	706
<b>54</b>	<b>LL BUS Generic Driver.....</b>	<b>730</b>
54.1	BUS Firmware driver API description.....	730
54.1.1	Detailed description of functions .....	730
54.2	BUS Firmware driver defines .....	747
54.2.1	BUS .....	747

<b>55</b>	<b>LL COMP Generic Driver.....</b>	<b>750</b>
55.1	COMP Firmware driver registers structures .....	750
55.1.1	LL_COMP_InitTypeDef .....	750
55.2	COMP Firmware driver API description .....	750
55.2.1	Detailed description of functions .....	750
55.3	COMP Firmware driver defines .....	759
55.3.1	COMP .....	759
<b>56</b>	<b>LL CORTEX Generic Driver.....</b>	<b>762</b>
56.1	CORTEX Firmware driver API description .....	762
56.1.1	Detailed description of functions .....	762
56.2	CORTEX Firmware driver defines.....	768
56.2.1	CORTEX.....	768
<b>57</b>	<b>LL CRC Generic Driver.....</b>	<b>771</b>
57.1	CRC Firmware driver API description .....	771
57.1.1	Detailed description of functions .....	771
57.2	CRC Firmware driver defines .....	777
57.2.1	CRC .....	777
<b>58</b>	<b>LL CRS Generic Driver.....</b>	<b>779</b>
58.1	CRS Firmware driver API description.....	779
58.1.1	Detailed description of functions .....	779
58.2	CRS Firmware driver defines .....	790
58.2.1	CRS .....	790
<b>59</b>	<b>LL DAC Generic Driver.....</b>	<b>793</b>
59.1	DAC Firmware driver registers structures .....	793
59.1.1	LL_DAC_InitTypeDef.....	793
59.2	DAC Firmware driver API description.....	793
59.2.1	Detailed description of functions .....	793
59.3	DAC Firmware driver defines .....	810
59.3.1	DAC .....	810
<b>60</b>	<b>LL DMA Generic Driver .....</b>	<b>816</b>
60.1	DMA Firmware driver registers structures .....	816
60.1.1	LL_DMA_InitTypeDef .....	816
60.2	DMA Firmware driver API description .....	817
60.2.1	Detailed description of functions .....	817
60.3	DMA Firmware driver defines.....	851

60.3.1	DMA.....	851
<b>61</b>	<b>LL EXTI Generic Driver .....</b>	<b>857</b>
61.1	EXTI Firmware driver registers structures .....	857
61.1.1	LL_EXTI_InitTypeDef .....	857
61.2	EXTI Firmware driver API description .....	857
61.2.1	Detailed description of functions .....	857
61.3	EXTI Firmware driver defines.....	872
61.3.1	EXTI.....	872
<b>62</b>	<b>LL GPIO Generic Driver .....</b>	<b>875</b>
62.1	GPIO Firmware driver registers structures .....	875
62.1.1	LL_GPIO_InitTypeDef .....	875
62.2	GPIO Firmware driver API description .....	875
62.2.1	Detailed description of functions .....	875
62.3	GPIO Firmware driver defines.....	890
62.3.1	GPIO.....	890
<b>63</b>	<b>LL I2C Generic Driver .....</b>	<b>893</b>
63.1	I2C Firmware driver registers structures .....	893
63.1.1	LL_I2C_InitTypeDef.....	893
63.2	I2C Firmware driver API description.....	894
63.2.1	Detailed description of functions .....	894
63.3	I2C Firmware driver defines .....	934
63.3.1	I2C .....	934
<b>64</b>	<b>LL I2S Generic Driver .....</b>	<b>939</b>
64.1	I2S Firmware driver registers structures .....	939
64.1.1	LL_I2S_InitTypeDef.....	939
64.2	I2S Firmware driver API description.....	940
64.2.1	Detailed description of functions .....	940
64.3	I2S Firmware driver defines .....	953
64.3.1	I2S .....	953
<b>65</b>	<b>LL IWDG Generic Driver.....</b>	<b>956</b>
65.1	IWDG Firmware driver API description .....	956
65.1.1	Detailed description of functions .....	956
65.2	IWDG Firmware driver defines .....	960
65.2.1	IWDG .....	960
<b>66</b>	<b>LL LPTIM Generic Driver.....</b>	<b>961</b>

66.1	LPTIM Firmware driver registers structures .....	961
66.1.1	LL_LPTIM_InitTypeDef.....	961
66.2	LPTIM Firmware driver API description.....	961
66.2.1	Detailed description of functions .....	961
66.3	LPTIM Firmware driver defines .....	984
66.3.1	LPTIM .....	984
<b>67</b>	<b>LL LPUART Generic Driver .....</b>	<b>988</b>
67.1	LPUART Firmware driver registers structures.....	988
67.1.1	LL_LPUART_InitTypeDef.....	988
67.2	LPUART Firmware driver API description .....	988
67.2.1	Detailed description of functions .....	988
67.3	LPUART Firmware driver defines.....	1026
67.3.1	LPUART .....	1026
<b>68</b>	<b>LL PWR Generic Driver .....</b>	<b>1030</b>
68.1	PWR Firmware driver API description.....	1030
68.1.1	Detailed description of functions .....	1030
68.2	PWR Firmware driver defines .....	1040
68.2.1	PWR .....	1040
<b>69</b>	<b>LL RCC Generic Driver .....</b>	<b>1043</b>
69.1	RCC Firmware driver registers structures .....	1043
69.1.1	LL_RCC_ClocksTypeDef .....	1043
69.2	RCC Firmware driver API description .....	1043
69.2.1	Detailed description of functions .....	1043
69.3	RCC Firmware driver defines .....	1078
69.3.1	RCC .....	1078
<b>70</b>	<b>LL RNG Generic Driver .....</b>	<b>1087</b>
70.1	RNG Firmware driver API description .....	1087
70.1.1	Detailed description of functions .....	1087
70.2	RNG Firmware driver defines.....	1090
70.2.1	RNG.....	1090
<b>71</b>	<b>LL RTC Generic Driver .....</b>	<b>1092</b>
71.1	RTC Firmware driver registers structures .....	1092
71.1.1	LL_RTC_InitTypeDef.....	1092
71.1.2	LL_RTC_TimeTypeDef.....	1092
71.1.3	LL_RTC_DateTypeDef.....	1093
71.1.4	LL_RTC_AlarmTypeDef .....	1093

71.2	RTC Firmware driver API description .....	1094
71.2.1	Detailed description of functions .....	1094
71.3	RTC Firmware driver defines .....	1160
71.3.1	RTC .....	1160
<b>72</b>	<b>LL SPI Generic Driver.....</b>	<b>1169</b>
72.1	SPI Firmware driver registers structures .....	1169
72.1.1	LL_SPI_InitTypeDef .....	1169
72.2	SPI Firmware driver API description .....	1170
72.2.1	Detailed description of functions .....	1170
72.3	SPI Firmware driver defines .....	1188
72.3.1	SPI .....	1188
<b>73</b>	<b>LL SYSTEM Generic Driver.....</b>	<b>1192</b>
73.1	SYSTEM Firmware driver API description .....	1192
73.1.1	Detailed description of functions .....	1192
73.2	SYSTEM Firmware driver defines .....	1208
73.2.1	SYSTEM .....	1208
<b>74</b>	<b>LL TIM Generic Driver .....</b>	<b>1211</b>
74.1	TIM Firmware driver registers structures .....	1211
74.1.1	LL_TIM_InitTypeDef .....	1211
74.1.2	LL_TIM_OC_InitTypeDef.....	1211
74.1.3	LL_TIM_IC_InitTypeDef .....	1212
74.1.4	LL_TIM_ENCODER_InitTypeDef .....	1212
74.2	TIM Firmware driver API description .....	1214
74.2.1	Detailed description of functions .....	1214
74.3	TIM Firmware driver defines.....	1264
74.3.1	TIM.....	1264
<b>75</b>	<b>LL USART Generic Driver .....</b>	<b>1275</b>
75.1	USART Firmware driver registers structures .....	1275
75.1.1	LL_USART_InitTypeDef .....	1275
75.1.2	LL_USART_ClockInitTypeDef.....	1275
75.2	USART Firmware driver API description .....	1276
75.2.1	Detailed description of functions .....	1276
75.3	USART Firmware driver defines.....	1344
75.3.1	USART.....	1344
<b>76</b>	<b>LL UTILS Generic Driver .....</b>	<b>1350</b>
76.1	UTILS Firmware driver registers structures.....	1350

76.1.1	LL_UTILS_PLLInitTypeDef .....	1350
76.1.2	LL_UTILS_ClkInitTypeDef.....	1350
76.2	UTILS Firmware driver API description .....	1351
76.2.1	System Configuration functions.....	1351
76.2.2	Detailed description of functions .....	1351
76.3	UTILS Firmware driver defines.....	1354
76.3.1	UTILS.....	1354
<b>77</b>	<b>LL WWDG Generic Driver .....</b>	<b>1355</b>
77.1	WWDG Firmware driver API description .....	1355
77.1.1	Detailed description of functions .....	1355
77.2	WWDG Firmware driver defines.....	1359
77.2.1	WWDG.....	1359
<b>78</b>	<b>Correspondence between API registers and API low-layer driver functions.....</b>	<b>1360</b>
78.1	ADC .....	1360
78.2	BUS.....	1365
78.3	COMP .....	1373
78.4	CORTEX .....	1374
78.5	CRC .....	1375
78.6	CRS .....	1376
78.7	DAC .....	1377
78.8	DMA .....	1379
78.9	EXTI.....	1383
78.10	GPIO .....	1383
78.11	I2C .....	1384
78.12	I2S.....	1388
78.13	IWDG .....	1390
78.14	LPTIM .....	1390
78.15	LPUART.....	1393
78.16	PWR.....	1397
78.17	RCC .....	1398
78.18	RNG .....	1402
78.19	RTC.....	1403
78.20	SPI .....	1412
78.21	SYSTEM .....	1414

78.22	TIM.....	1416
78.23	USART.....	1424
78.24	WWDG.....	1431
<b>79</b>	<b>FAQs.....</b>	<b>1432</b>
<b>80</b>	<b>Revision history .....</b>	<b>1436</b>

## List of tables

Table 1: Acronyms and definitions.....	25
Table 2: HAL drivers files.....	27
Table 3: User-application files .....	28
Table 4: APis classification .....	33
Table 5: List of devices supported by HAL drivers .....	34
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 .....	46
Table 12: Description of GPIO_InitTypeDef structure .....	48
Table 13: Description of EXTI configuration macros .....	51
Table 14: MSP functions.....	55
Table 15: Timeout values .....	59
Table 16: LL drivers files.....	63
Table 17: Common peripheral initialization functions .....	66
Table 18: Optional peripheral initialization functions .....	66
Table 19: Specific Interrupt, DMA request and status flags management .....	68
Table 20: Available function formats.....	68
Table 21: Peripheral clock activation/deactivation management .....	69
Table 22: Peripheral activation/deactivation management.....	69
Table 23: Peripheral configuration management.....	69
Table 24: Peripheral register management .....	69
Table 25: Correspondence between ADC registers and ADC low-layer driver functions .....	1360
Table 26: Correspondence between BUS registers and BUS low-layer driver functions.....	1365
Table 27: Correspondence between COMP registers and COMP low-layer driver functions.....	1373
Table 28: Correspondence between CORTEX registers and CORTEX low-layer driver functions .....	1374
Table 29: Correspondence between CRC registers and CRC low-layer driver functions.....	1375
Table 30: Correspondence between CRS registers and CRS low-layer driver functions .....	1376
Table 31: Correspondence between DAC registers and DAC low-layer driver functions .....	1377
Table 32: Correspondence between DMA registers and DMA low-layer driver functions .....	1379
Table 33: Correspondence between EXTI registers and EXTI low-layer driver functions .....	1383
Table 34: Correspondence between GPIO registers and GPIO low-layer driver functions .....	1383
Table 35: Correspondence between I2C registers and I2C low-layer driver functions .....	1384
Table 36: Correspondence between I2S registers and I2S low-layer driver functions.....	1388
Table 37: Correspondence between IWDG registers and IWDG low-layer driver functions .....	1390
Table 38: Correspondence between LPTIM registers and LPTIM low-layer driver functions .....	1390
Table 39: Correspondence between LPUART registers and LPUART low-layer driver functions .....	1393
Table 40: Correspondence between PWR registers and PWR low-layer driver functions.....	1397
Table 41: Correspondence between RCC registers and RCC low-layer driver functions .....	1398
Table 42: Correspondence between RNG registers and RNG low-layer driver functions .....	1402
Table 43: Correspondence between RTC registers and RTC low-layer driver functions .....	1403
Table 44: Correspondence between SPI registers and SPI low-layer driver functions .....	1412
Table 45: Correspondence between SYSTEM registers and SYSTEM low-layer driver functions.....	1414
Table 46: Correspondence between TIM registers and TIM low-layer driver functions .....	1416
Table 47: Correspondence between USART registers and USART low-layer driver functions .....	1424
Table 48: Correspondence between WWDG registers and WWDG low-layer driver functions .....	1431
Table 49: Document revision history .....	1436

## List of figures

Figure 1: Example of project template .....	30
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
COMP	Comparator
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
LPTIM	Low Power Timer
MSP	MCU Specific Package
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer

<b>Acronym</b>	<b>Definition</b>
TSC	Touch Sensing Controller
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

## 2 Overview of HAL drivers

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

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

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

The HAL main features are the following:

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

### 2.1 HAL and user-application files

#### 2.1.1 HAL driver files

HAL drivers are composed of the following set of files:

**Table 2: HAL drivers files**

File	Description
<i>stm32l0xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32l0xx_hal_adc.c, stm32l0xx_hal_irda.c, ...</i>
<i>stm32l0xx_hal_ppp.h</i>	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. <i>Example: stm32l0xx_hal_adc.h, stm32l0xx_hal_irda.h, ...</i>

File	Description
<i>stm32l0xx_hal_ppp_ex.c</i>	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. <i>Example: stm32l0xx_hal_adc_ex.c, stm32l0xx_hal_dma_ex.c, ...</i>
<i>stm32l0xx_hal_ppp_ex.h</i>	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs <i>Example: stm32l0xx_hal_adc_ex.h, stm32l0xx_hal_dma_ex.h, ...</i>
<i>stm32l0xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32l0xx_hal.h</i>	<i>stm32l0xx_hal.c</i> header file
<i>stm32l0xx_hal_msp_template.c</i>	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l0xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32l0xx_hal_def.h</i>	Common HAL resources such as common define statements, enumerations, structures and macros.

## 2.1.2 User-application files

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

Table 3: User-application files

File	Description
<i>system_stm32l0xx.c</i>	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows to relocate the vector table in internal SRAM.
<i>startup_stm32l0xx.s</i>	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
<i>stm32l0xx_flash.icf (optional)</i>	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32l0xx_FLASH.Id (optional)</i>	Linker file for SW4STM32 toolchain.
<i>stm32l0xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l0xx_hal_conf.h</i>	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
<i>stm32l0xx_it.c/h</i>	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in <i>stm32l0xx_hal.c</i> ) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
<i>main.c/h</i>	This file contains the main program routine, in particular call to HAL_Init(), assert_failed() implementation, system clock configuration, peripheral HAL initialization and user application code.

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

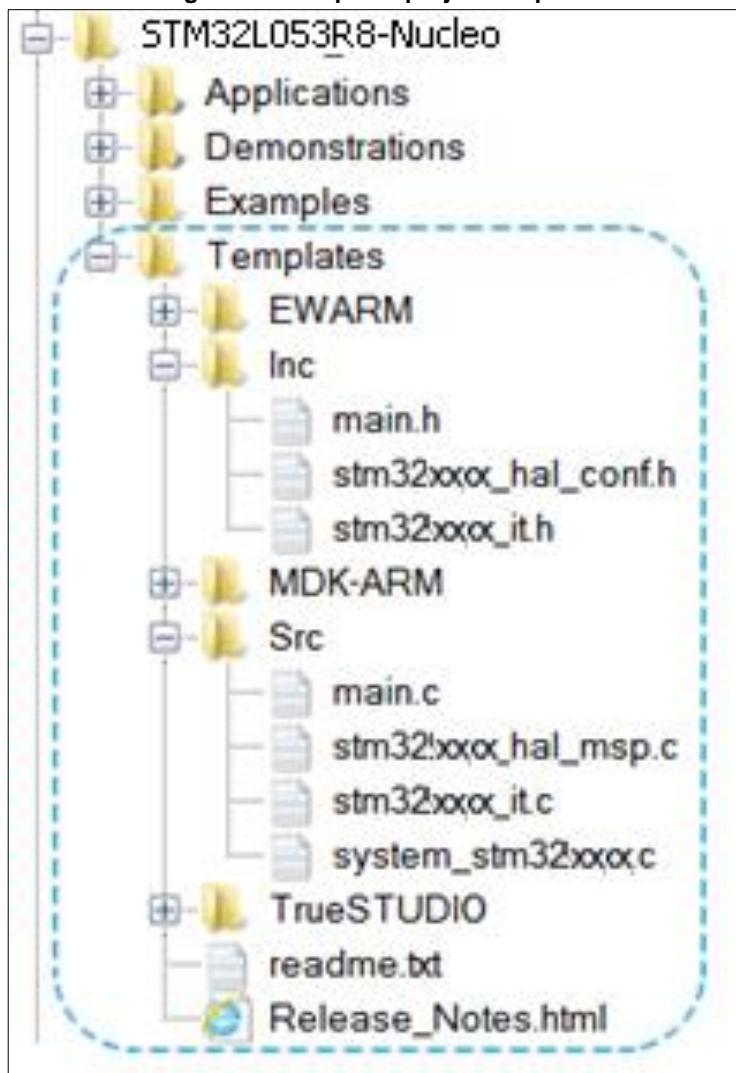
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

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



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

Figure 1: Example of project template



## 2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

### 2.2.1 Peripheral handle structures

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

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

The peripheral handle is used for the following purposes:

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

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

An example of peripheral structure is shown below:

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



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

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



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



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

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

## 2.2.2 Initialization and configuration structure

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

```
typedef struct
{
    uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
    uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
    in a frame.*/
    uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
    uint32_t Parity; /*!< Specifies the parity mode. */
    uint32_t Mode; /*!< Specifies whether the Receive or Transmit mode is enabled or
    disabled.*/
    uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled
    or disabled.*/
    uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or
    disabled,
    to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```



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

```
HAL ADC ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef*
sConfig)
```

## 2.2.3 Specific process structures

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

Example:

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

## 2.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL drivers files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_DeInit(ADC_HandleTypeDef *hadc); HAL_StatusTypeDef
HAL_ADC_Start(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_Stop(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc); HAL_StatusTypeDef
HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc); void HAL_ADC_IRQHandler(ADC_HandleTypeDef*
hadc);
```

- **Extension APIs:** This set of API is divided into two sub-categories :
  - **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_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(STM32L051xx) && !defined(STM32L061xx) void
HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit); void
```

```

HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void); RCC_CRSStatusTypeDef
HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout); #endif /* !(STM32L051xx) &&
!(STM32L061xx) */ The data structure related to the specific APIs is delimited by
the device part number define statement. It is located in the corresponding
extension header C file.

```

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

**Table 4: APIs classification**

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

**Notes:**

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



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



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

## 2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal.c	Yes															
stm32l0xx_hal_adc.c	Yes															
stm32l0xx_hal_adc_ex.c	Yes															
stm32l0xx_hal_comp.c	Yes															
stm32l0xx_hal_cortex.c	Yes															
stm32l0xx_hal_crc.c	Yes															
stm32l0xx_hal_crc_ex.c	Yes															
stm32l0xx_hal_cryp.c	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_cryp_ex.c	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_dac.c	No	No	No	No	No	Yes	Yes									
stm32l0xx_hal_dac_ex.c	No	No	No	No	No	Yes	Yes									
stm32l0xx_hal_dma.c	Yes															
stm32l0xx_hal_firewall.c	No	No	No	No	Yes											
stm32l0xx_hal_flash.c	Yes															

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_flash_ex.c	Yes															
stm32l0xx_hal_flash_ramfunc.c	Yes															
stm32l0xx_hal_gpio.c	Yes															
stm32l0xx_hal_i2c.c	Yes															
stm32l0xx_hal_i2c_ex.c	Yes															
stm32l0xx_hal_i2s.c	No	No	No	No	Yes											
stm32l0xx_hal_irda.c	Yes															
stm32l0xx_hal_iwdg.c	Yes															
stm32l0xx_hal_lcd.c	No	No	No	No	No	No	Yes									
stm32l0xx_hal_lptim.c	Yes															
stm32l0xx_hal_pcd.c	No	No	No	No	No	Yes	Yes									
stm32l0xx_hal_pcd_ex.c	No	No	No	No	No	Yes	Yes									
stm32l0xx_hal_pwr.c	Yes															
stm32l0xx_hal_pwr_ex.c	Yes															

## Overview of HAL drivers

UM1749

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
<code>stm32l0xx_hal_rcc.c</code>	Yes															
<code>stm32l0xx_hal_rcc_ex.c</code>	Yes															
<code>stm32l0xx_hal_rng.c</code>	No	No	No	No	No	Yes	Yes									
<code>stm32l0xx_hal_rtc.c</code>	Yes															
<code>stm32l0xx_hal_rtc_ex.c</code>	Yes															
<code>stm32l0xx_hal_smbus.c</code>	Yes															
<code>stm32l0xx_hal_smartcard.c</code>	Yes															
<code>stm32l0xx_hal_smartcard_ex.c</code>	Yes															
<code>stm32l0xx_hal_spi.c</code>	Yes															
<code>stm32l0xx_hal_tim.c</code>	Yes															
<code>stm32l0xx_hal_tim_ex.c</code>	Yes															
<code>stm32l0xx_hal_tsc.c</code>	No	No	No	No	No	Yes	Yes									
<code>stm32l0xx_hal_uart.c</code>	Yes															
<code>stm32l0xx_hal_uart_ex.c</code>	Yes															

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_usart.c	Yes															
stm32l0xx_hal_wwdg.c	Yes															

## 2.5 HAL drivers rules

### 2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

**Table 6: HAL API naming rules**

	Generic	Family specific	Device specific
File names	<i>stm32l0xx_hal_ppp (c/h)</i>	<i>stm32l0xx_hal_ppp_ex (c/h)</i>	<i>stm32l0xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with \_TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32L0xx reference manuals.
- Peripheral registers are declared in the PPP\_TypeDef structure (e.g. ADC\_TypeDef) in stm32l0xxx.h header file. stm32l0xxx.h corresponds to stm32l051xx.h, stm32l052xx.h, stm32l053xx.h, stm32l061xx.h, stm32l062xx.h or stm32l063xx.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:  
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 */
}
```

- GPIO
- SYSTICK
- NVIC
- RCC
- FLASH.

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

**Table 7: Macros handling interrupts and specific clock configurations**

Macros	Description
<code>_HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Enables a specific peripheral interrupt
<code>_HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Disables a specific peripheral interrupt
<code>_HAL_PPP_GET_IT (__HANDLE__, __INTERRUPT__)</code>	Gets a specific peripheral interrupt status
<code>_HAL_PPP_CLEAR_IT (__HANDLE__, __INTERRUPT__)</code>	Clears a specific peripheral interrupt status
<code>_HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)</code>	Gets a specific peripheral flag status
<code>_HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)</code>	Clears a specific peripheral flag status
<code>_HAL_PPP_ENABLE(__HANDLE__)</code>	Enables a peripheral
<code>_HAL_PPP_DISABLE(__HANDLE__)</code>	Disables a peripheral
<code>_HAL_PPP_XXXX (__HANDLE__, __PARAM__)</code>	Specific PPP HAL driver macro
<code>_HAL_PPP_GET_IT_SOURCE (__HANDLE__, __INTERRUPT__)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the `stm320xx_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 stm32l0xx\_it.c
- User callback functions.

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

There are three types of user callbacks functions:

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

**Table 8: Callback functions**

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

## 2.6 HAL generic APIs

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

- **Initialization and de-initialization functions:** HAL\_PPP\_Init(), HAL\_PPP\_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 timeout 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
	<i>HAL_ADC_ConvCpltCallback()</i>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	<i>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
<i>Control group</i>	<i>HAL_ADC_ConfigChannel()</i>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	<i>HAL_ADC_AnalogWDGConfig</i>	This function configures the analog watchdog for the selected ADC
<i>State and Errors group</i>	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function allows getting in runtime the error that occurred during IT routine

## 2.7 HAL extension APIs

### 2.7.1 HAL extension model overview

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

The extension model consists of an additional file, `stm32l0xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32l0xx_hal_ppp_ex.h`) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
<i>HAL_ADCEx_Calibration_Start()</i>	This function is used to start the automatic ADC calibration
<i>HAL_ADCEx_Calibration_GetValue()</i>	This function is used to get the ADC calibration factor
<i>HAL_ADCEx_Calibration_SetValue()</i>	This function is used to set the calibration factor to overwrite automatic conversion result

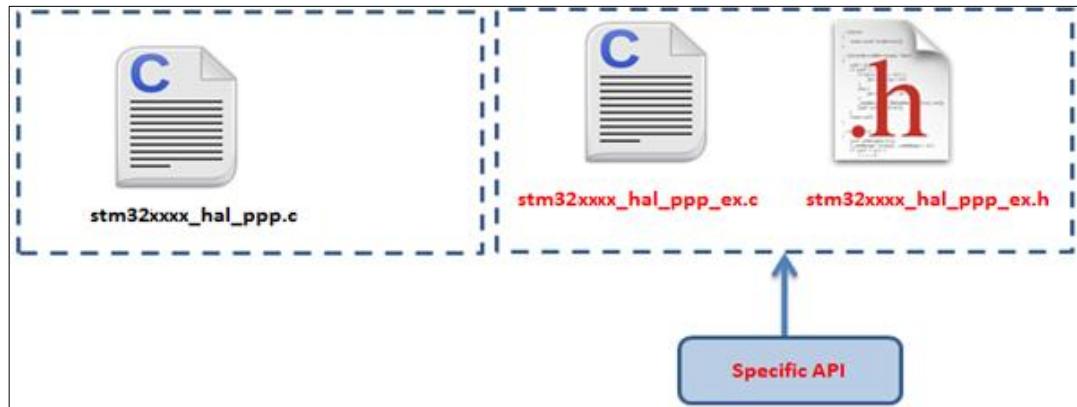
## 2.7.2 HAL extension model cases

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

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

When a new feature specific to a given device is required, the new APIs are added in the `stm32l0xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEEx_Function()`.

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



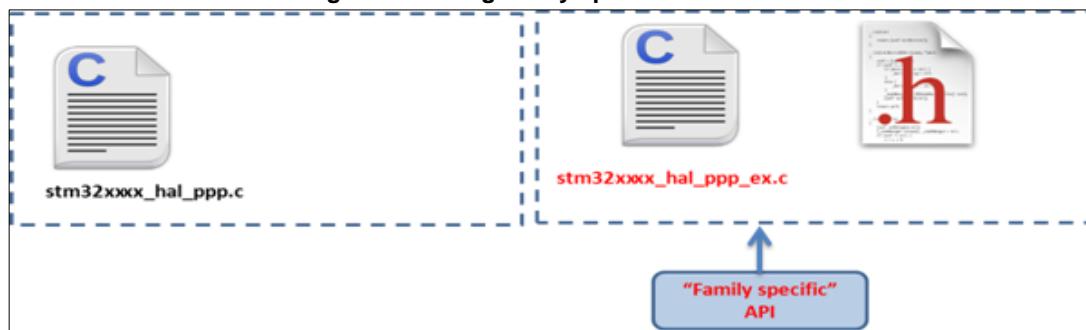
Example: `stm32l0xx_hal_rcc_ex.c/h`

```
#if !defined(STM32L051xx) && !defined(STM32L061xx)
void HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit);
void HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void);
void HAL_RCCEx CRSGetSynchronizationInfo(RCC_CRSSynchroInfoTypeDef *pSynchroInfo);
RCC_CRSStatusTypeDef HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout);
#endif /* !(STM32L051xx) && !(STM32L061xx) */
```

### Case2: Adding a family-specific function

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

**Figure 3: Adding family-specific functions**

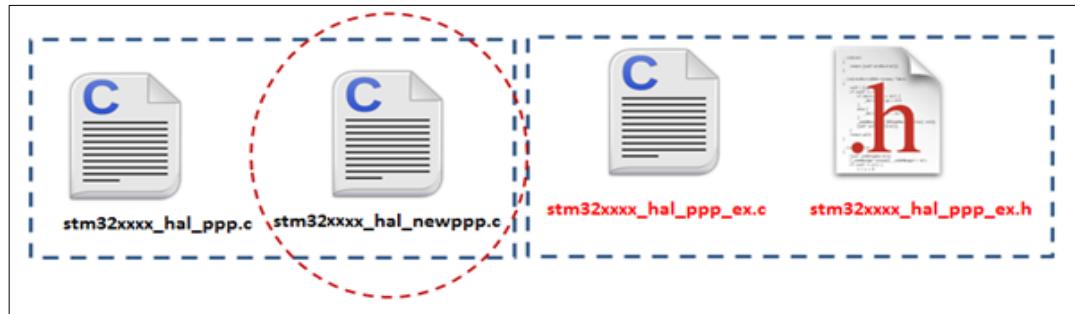


### Case3: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32l0xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

**Figure 4: Adding new peripherals**

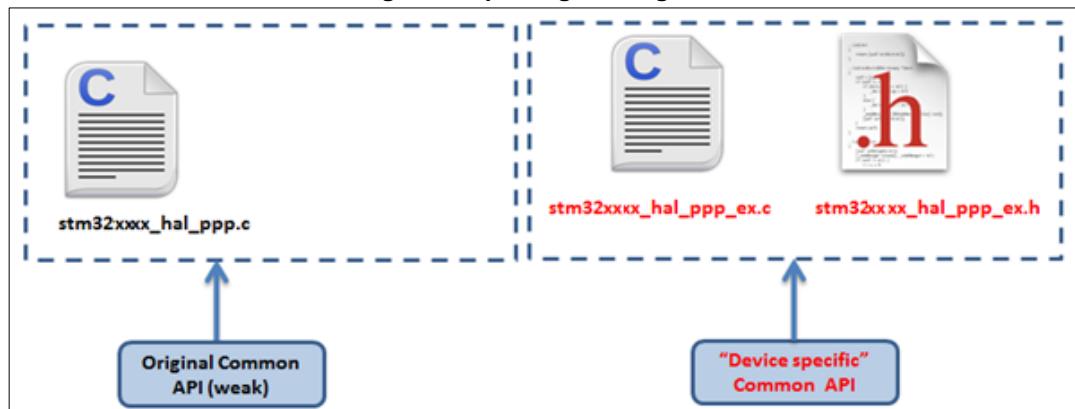


Example: `stm32l0xx_hal_lcd.c/h`

### Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32l0xx_hal_ppp_ex.c` extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

**Figure 5: Updating existing APIs**



### Case5 : Updating existing data structures

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

Example:

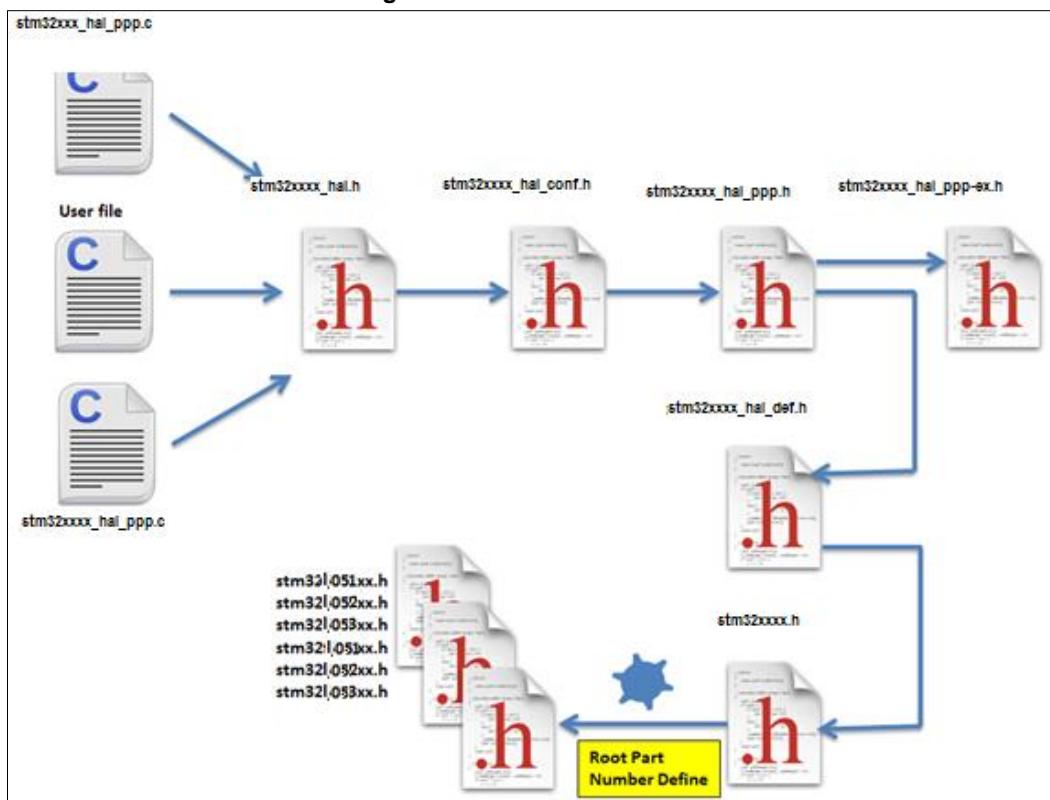
```
#if defined (STM32L051xx)
typedef struct
{
(...)

}PPP_InitTypeDef;
#endif /* STM32L051xx */
```

## 2.8 File inclusion model

The header of the common HAL driver file (stm32l0xx\_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 stm32l0xx_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 *stm32l0xx\_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 *stm32l0xx\_hal\_def.h* file calls the *stm32l0xx.h* file in CMSIS library to get the data structures and the address mapping for all peripherals:

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

- **Common macros**

- Macro defining HAL\_MAX\_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
        – Macro linking a PPP peripheral to a DMA structure pointer:__HAL_LINKDMA();
```

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

## 2.10 HAL configuration

The configuration file, *stm32l0xx\_hal\_conf.h*, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

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

Configuration item	Description	Default Value
<b>HSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)
<b>HSE_STARTUP_TIMEOUT</b>	Timeout for HSE start up, expressed in ms	5000

Configuration item	Description	Default Value
<b>HSI_VALUE</b>	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
<b>MSI_VALUE</b>	Defines the Internal Multiple Speed oscillator (MSI) value expressed in Hz.	2 000 000 (Hz)
<b>VDD_VALUE</b>	VDD value	3300 (mV)
<b>USE_RTOS</b>	Enables the use of RTOS	FALSE (for future use)
<b>PREFETCH_ENABLE</b>	Enables prefetch feature	TRUE
<b>BUFFER_CACHE_ENABLE</b>	Enables buffer cache	FALSE



The `stm32l0xx_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 `stm32l0xx_hal_conf_template.h` file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

## 2.11 HAL system peripheral handling

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

### 2.11.1 Clock

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

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

Some peripheral clocks are not derived from the system clock (RTC, USB...). In this case, the clock configuration is performed by an extended API defined in `stm32l0xx_hal_ppp_ex.c`: `HAL_RCCEEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)`.

Additional RCC HAL driver functions are available:

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

A set of macros are defined in `stm32l0xx_hal_rcc.h`. They allows executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

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

## 2.11.2 GPIOs

GPIO HAL APIs are the following:

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

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

When selecting EXTI mode with interrupt generation, the user must call `HAL_GPIO_EXTI_IRQHandler()` from `stm32l0xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

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

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

Structure field	Description
Speed	<p>Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ VERY_HIGH</p>
Alternate	<p>Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PP, where AFx: is the alternate function index PP: is the peripheral instance Example: use GPIO_AF1_TIM1 to connect TIM1 IOs on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines.</p> <p> Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.</p>

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external 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_FREQ_HIGH; HAL_GPIO_Init(GPIOA,
&GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

- Configuring USART1 Tx (PA9, mapped on AF4) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF4_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

### 2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, `stm32l0xx_hal_cortex.c`, provides APIs to handle NVIC and Systick. The supported APIs include:

- `HAL_NVIC_SetPriority()`
- `HAL_NVIC_EnableIRQ()`/`HAL_NVIC_DisableIRQ()`
- `HAL_NVIC_SystemReset()`
- `HAL_SYSTICK_IRQHandler()`
- `HAL_NVIC_GetPendingIRQ()` / `HAL_NVIC_SetPendingIRQ()` / `HAL_NVIC_ClearPendingIRQ()`
- `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_PVDConfig()`
  - `HAL_PWR_EnablePVD()` / `HAL_PWR_DisablePVD()`
  - `HAL_PWR_PVD_IRQHandler()`
  - `HAL_PWR_PVDCallback()`
- Wakeup pin configuration
  - `HAL_PWR_EnableWakeUpPin()` / `HAL_PWR_DisableWakeUpPin()`
- Low power mode entry
  - `HAL_PWR_EnterSLEEPMode()`
  - `HAL_PWR_EnterSTOPMode()`
  - `HAL_PWR_EnterSTANDBYMode()`

Depending on the STM32 Series, extension functions are available in `stm32l0xx_hal_pwr_ex`. Here are a few examples (the list is not exhaustive)

- Ultra low power mode control
  - `HAL_PWREx_EnableUltraLowPower()` / `HAL_PWREx_DisableUltraLowPower()`
  - `HAL_PWREx_EnableLowPowerRunMode()` / `HAL_PWREx_DisableLowPowerRunMode()`

### 2.11.5 EXTI

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

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

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

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

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

If the EXTI interrupt mode is selected, the user application must call `HAL_PPP_FUNCTION_IRQHandler()` (for example `HAL_PWR_PVD_IRQHandler()`), from `stm32l0xx_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 stream, `HAL_DMA_Init()` API allows programming the required configuration through the following parameters:

- Transfer Direction
- Source and Destination data formats
- Circular, Normal or peripheral flow control mode
- Channels Priority level
- Source and Destination Increment mode
- FIFO mode and its Threshold (if needed)
- Burst mode for Source and/or Destination (if needed).

Two operating modes are available:

- Polling mode I/O operation

- a. Use HAL\_DMA\_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
- b. Use HAL\_DMA\_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
  - a. Configure the DMA interrupt priority using HAL\_NVIC\_SetPriority()
  - b. Enable the DMA IRQ handler using HAL\_NVIC\_EnableIRQ()
  - c. Use HAL\_DMA\_Start\_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
  - d. Use HAL\_DMA\_IRQHandler() called under DMA\_IRQHandler() Interrupt subroutine
  - e. When data transfer is complete, HAL\_DMA\_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

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

- Use HAL\_DMA\_GetState() function to return the DMA state and HAL\_DMA\_GetError() in case of error detection.
- Use HAL\_DMA\_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- \_\_HAL\_DMA\_ENABLE: enables the specified DMA Channels.
- \_\_HAL\_DMA\_DISABLE: disables the specified DMA Channels.
- \_\_HAL\_DMA\_GET\_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”).



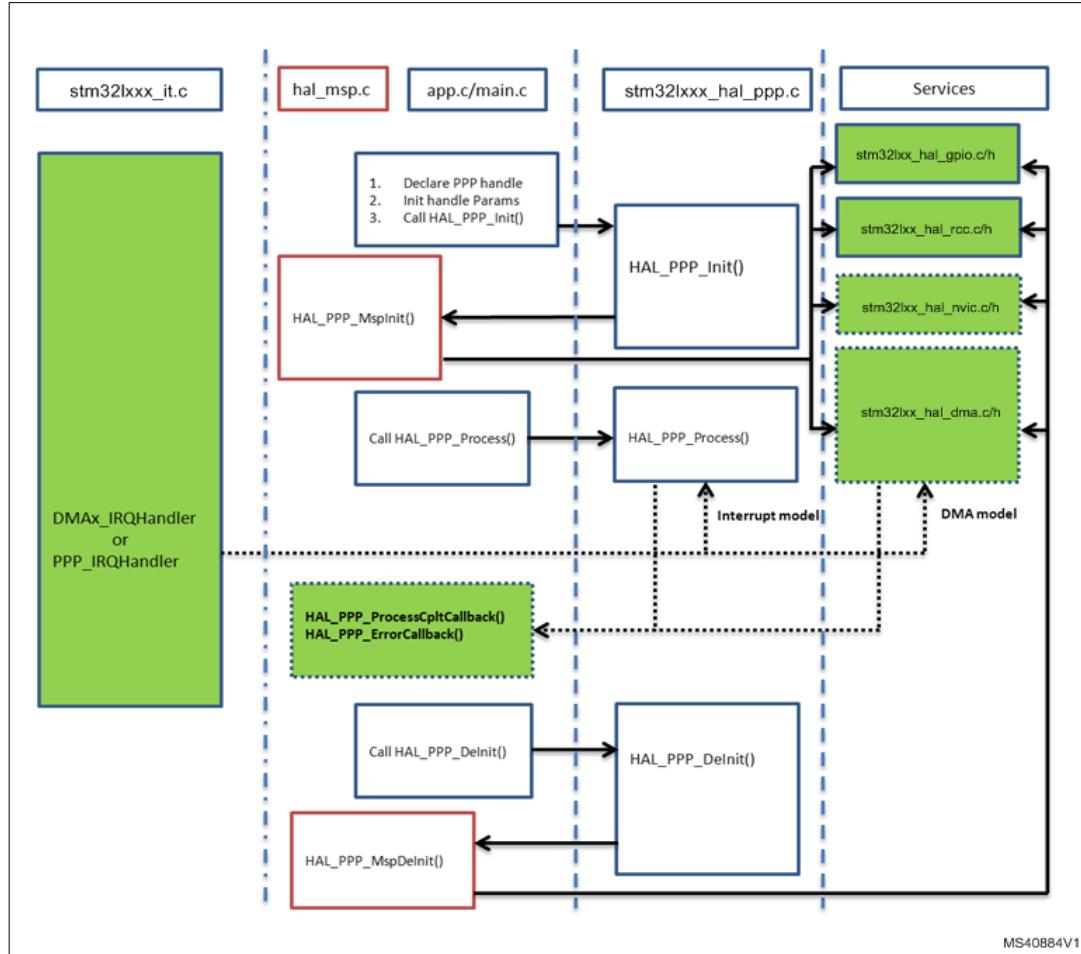
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 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



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 (see green blocks in the above schematics).

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 (see blue and red blocks in the above schematics).

## 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 `stm32l0xx_hal.c`.

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

### 2.12.2.2 System clock initialization

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

```
void SystemClock_Config(void)
{
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_OscInitTypeDef RCC_OscInitStruct = {0};

/* Enable MSI Oscillator */
RCC_OscInitStruct.OscillatorType = RCC OSCILLATORTYPE_MSI;
RCC_OscInitStruct.MSISState = RCC MSI_ON;
RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
RCC_OscInitStruct.MSICalibrationValue=0x00;
RCC_OscInitStruct.PLL.PLLState = RCC PLL NONE;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    /* Initialization Error */
    while(1);
}

/* Select MSI as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks
dividers */
RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
    /* Initialization Error */
    while(1);
}
```

```

    }

    /* Enable Power Control clock */
    __HAL_RCC_PWR_CLK_ENABLE();
    /* The voltage scaling allows optimizing the power consumption when the device is
     * clocked below the
     * maximum system frequency, to update the voltage scaling value regarding system
     * frequency refer to product datasheet. */
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
    /* Disable Power Control clock      */
    HAL_RCC_PWR_CLK_DISABLE();
}

```

### 2.12.2.3 HAL MSP initialization process

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

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

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

```

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

```

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

The HAL MSP callback is implemented inside the `stm32l0xx_hal_msp.c` file in the user folders. An `stm32l0xx_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.

`stm32l0xx_hal_msp.c` file contains the following functions:

Table 14: MSP functions

Routine	Description
<code>void HAL_MspInit()</code>	Global MSP initialization routine
<code>void HAL_MspDeInit()</code>	Global MSP de-initialization routine
<code>void HAL_PPP_MspInit()</code>	PPP MSP initialization routine
<code>void HAL_PPP_MspDeInit()</code>	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 `stm32l0xx_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)
{}
```

*stm32l0xx\_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 `stm32l0xx_it.c`. When DMA mode is used, the DMA initialization is done in the `HAL_PPP_MspInit()` callback. The user should

also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

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

The initialization is done as follows (UART example):

```
int main(void)
{
    /* Set User Parameters */
    UartHandle.Init.BaudRate = 9600;
    UartHandle.Init.WordLength = UART_DATABITS_8;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
    UartHandle.Init.Parity = UART_PARITY_NONE;
    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode = UART_MODE_TX_RX;
    UartHandle.Init.Instance = USART1;
    HAL_USART_Init(&UartHandle);
    ...
}
void HAL_USART_MspInit (USART_HandleTypeDef * huart)
{
    static DMA_HandleTypeDef hdma_tx;
    static DMA_HandleTypeDef hdma_rx;
    ...
    HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
    HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
    ...
}
```

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

An example of use is illustrated below:

*main.c* file:

```
USART_HandleTypeDef UartHandle;
int main(void)
{
    /* Set User Parameters */
    UartHandle.Init.BaudRate = 9600;
    UartHandle.Init.WordLength = UART_DATABITS_8;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
    UartHandle.Init.Parity = UART_PARITY_NONE;
    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
    HAL_USART_Init(&UartHandle);
    HAL_USART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
    while (1);
}
void HAL_USART_TxCpltCallback(USART_HandleTypeDef *phuart)
{
}
void HAL_USART_TxErrorCallback(USART_HandleTypeDef *phuart)
{
}
```

*stm32l0xx\_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_USART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_USART_ErrorCallback ;
(...)
```

## 2.12.4 Timeout and error management

### 2.12.4.1 Timeout management

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

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

The timeout possible value are the following:

Table 15: Timeout values

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

**Notes:**

<sup>(1)</sup>*HAL\_MAX\_DELAY* is defined in the *stm32l0xx\_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)
{
    uint32_t tickstart = 0;
    ...
    tickstart = HAL_GetTick();
    ...
    while(ProcessOngoing)
    {
        ...
        if((HAL_GetTick() - tickstart) > LOCAL_PROCESS_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)
{
    uint32_t tickstart = 0;
    ...
    tickstart = HAL_GetTick();
    ...
    while(ProcessOngoing)
    {
        ...
        if(Timeout != HAL_MAX_DELAY)
        {
            if((HAL_GetTick() - tickstart) > Timeout)
            {
                /* Process unlocked */
                __HAL_UNLOCK(hppp);
                hppp->State= HAL PPP STATE TIMEOUT;
                return hppp->State;
            }
        }
        ...
    }
}
```

## 2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

```

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

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

```

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

- Timeout error: the following statement is used when a timeout error occurs: while (Process ongoing) { timeout = HAL\_GetTick() + Timeout; while (data processing is running) { if(timeout) { return HAL\_TIMEOUT; } } }

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

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

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

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

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

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

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

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

#### 2.12.4.3 Run-time checking

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

To enable the run-time checking, use the *assert\_param* macro, and leave the define **USE\_FULL\_ASSERT** uncommented in *stm32l0xx\_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 *stm32l0xx\_hal\_conf.h*:

```
/* Exported macro -----
#ifndef USE_FULL_ASSERT
/**
```

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

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

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



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

### 3 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 USB).

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 drivers files**

File	Description
<i>stm32l0xx_ll_bus.h</i>	This is the h-source file for core bus control and peripheral clock activation and deactivation <i>Example: LL_AHB1_GRP1_EnableClock</i>
<i>stm32l0xx_ll_ppp.h/c</i>	<i>stm32l0xx_ll_ppp.c</i> provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DelInit(). All the other APIs are defined within <i>stm32l0xx_ll_ppp.h</i> file. The Low Layer PPP driver is a standalone module. To use it, the application must include it in the <i>xx_ll_ppp.h</i> file.
<i>stm32l0xx_ll_cortex.h</i>	CortexM related register operation APIs including the Systick, Low power (LL_SYSTICK_xxxxx, LL_LPM_xxxxx "Low Power Mode" ...)
<i>stm32l0xx_ll_utils.h/c</i>	This file covers the generic APIs: <ul style="list-style-type: none"> <li>• Read of device unique ID and electronic signature</li> <li>• Timebase and delay management</li> <li>• System clock configuration.</li> </ul>
<i>stm32l0xx_ll_system.h</i>	System related operations (LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx)

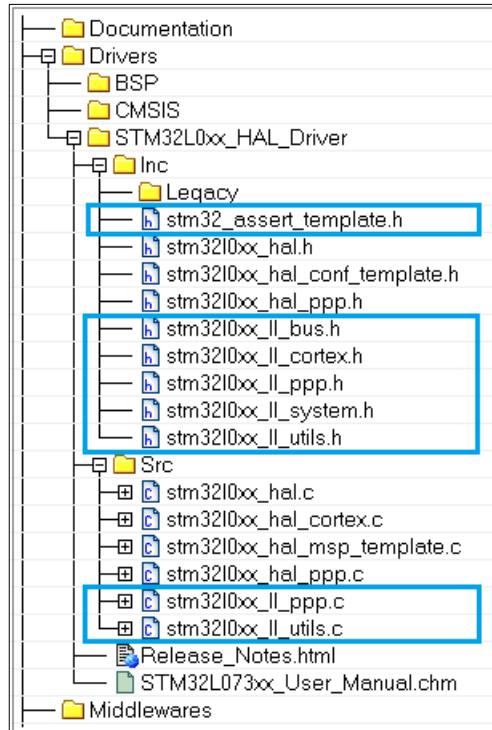
File	Description
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and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



There is no configuration file for the LL drivers.

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

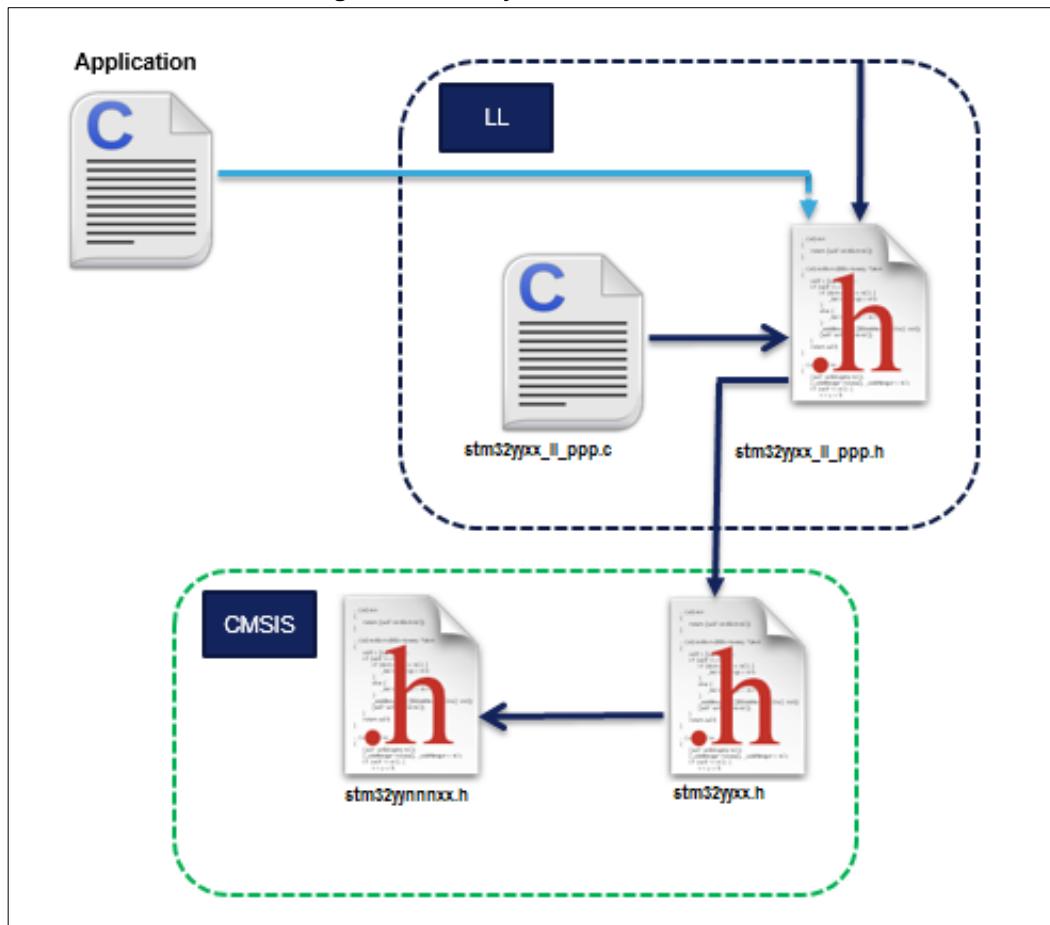
**Figure 8: Low Layer driver folders**



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 drivers 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 `stm32l0xx_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	<ul style="list-style-type: none"> <li>• <i>PPP_TypeDef*</i> <i>PPPx</i></li> <li>• <i>LL_PPP_InitTypeDef*</i> <i>PPP_InitStruct</i></li> </ul>	<p>Initializes the peripheral main features according to the parameters specified in <i>PPP_InitStruct</i>.</p> <p>Example:  <code>LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)</code></p>
LL_PPP_StructInit	void	<ul style="list-style-type: none"> <li>• <i>LL_PPP_InitTypeDef*</i> <i>PPP_InitStruct</i></li> </ul>	<p>Fills each <i>PPP_InitStruct</i> member with its default value.</p> <p>Example.  <code>LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)</code></p>
LL_PPP_Delinit	ErrorStatus	<ul style="list-style-type: none"> <li>• <i>PPP_TypeDef*</i> <i>PPPx</i></li> </ul>	<p>De-initializes the peripheral registers, that is restore them to their default reset values.</p> <p>Example.  <code>LL_USART_Delinit(USART_TypeDef *USARTx)</code></p>

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	ErrorStatus	<ul style="list-style-type: none"> <li>• <i>PPP_TypeDef*</i> <i>PPPx</i></li> <li>• <i>LL_PPP{CATEGORY}_InitTypeDef*</i> <i>PPP{CATEGORY}_InitStruct</i></li> </ul>	<p>Initializes peripheral features according to the parameters specified in <i>PPP_InitStruct</i>.</p> <p>Example:  <code>LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct)</code></p> <p><code>LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct)</code></p> <p><code>LL_TIM_IC_Init(TIM_TypeDef *TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef *TIM_IC_InitStruct)</code></p> <p><code>LL_TIM_ENCODER_Init(TIM_TypeDef *TIMx, LL_TIM_ENCODER_InitTypeDef *TIM_EncoderInitStruct)</code></p>

Functions	Return Type	Parameters	Examples
LL_PPP_{CATEGORY}_StructInit	void	<i>LL_PPP_{CATEGORY}_InitTypeDef*</i> <i>PPP_{CATEGORY}_InitStruct</i>	Fills each <i>PPP_{CATEGORY}_InitStruct</i> member with its default value.  Example: LL_RTC_TIME_StructInit(LL_RTC_TimeTypeDef *RTC_TimeStruct);
LL_PPP_CommonInit	ErrorStatus	<ul style="list-style-type: none"> <li>• <i>PPP_TypeDef*</i> <i>PPPx</i></li> <li>• <i>LL_PPP_CommonInitTypeDef*</i> <i>PPP_CommonInitStruct</i></li> </ul>	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)
LL_PPP_CommmonStructInit	void	<i>LL_PPP_CommonInitTypeDef*</i> <i>PPP_CommonInitStruct</i>	Fills each <i>PPP_CommonInitStruct</i> member with its default value  Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	ErrorStatus	<ul style="list-style-type: none"> <li>• <i>PPP_TypeDef*</i> <i>PPPx</i></li> <li>• <i>LL_PPP_ClockInitTypeDef*</i> <i>PPP_ClockInitStruct</i></li> </ul>	Initializes the peripheral clock configuration in synchronous mode.  Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	void	<i>LL_PPP_ClockInitTypeDef*</i> <i>PPP_ClockInitStruct</i>	Fills each <i>PPP_ClockInitStruct</i> 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 drivers functions. For more details please refer to [Section 4.12.4.3: "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
<i>LL_PPP_{_CATEGORY}_ActionItem_BITNAME</i>	<ul style="list-style-type: none"> <li>• LL_RCC_IsActiveFlag_LSIRDY</li> <li>• LL_RCC_IsActiveFlag_FWRST()</li> <li>• LL_ADC_ClearFlag_EOC(ADC1)</li> <li>• LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)</li> </ul>
<i>LL_PPP{_CATEGORY}_IsItem_BITNAME_Action</i>	

**Table 20: Available function formats**

Item	Action	Format
Flag	Get	<i>LL_PPP_IsActiveFlag_BITNAME</i>
	Clear	<i>LL_PPP_ClearFlag_BITNAME</i>
Interrupts	Enable	<i>LL_PPP_EnableIT_BITNAME</i>
	Disable	<i>LL_PPP_DisableIT_BITNAME</i>
	Get	<i>LL_PPP_IsEnabledIT_BITNAME</i>
DMA	Enable	<i>LL_PPP_EnableDMAReq_BITNAME</i>
	Disable	<i>LL_PPP_DisableDMAReq_BITNAME</i>
	Get	<i>LL_PPP_IsEnabledDMAReq_BITNAME</i>



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
<code>LL_bus_GRPx_ActionClock{Mode}</code>	<ul style="list-style-type: none"> <li>• <code>LL_IOP_GRP1_EnableClock (LL_IOP_GRP1_PERIPH_GPIOA LL_IOP_GRP1_PERIPH_GPIOB)</code></li> <li>• <code>LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)</code></li> </ul>



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



'bus' refers to the bus name (eg APB1).

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

**Table 22: Peripheral activation/deactivation management**

Name	Examples
<code>LL_PPP_{CATEGORY}_Action{Item}</code> <code>LL_PPP_{CATEGORY}_IsItemAction</code>	<ul style="list-style-type: none"> <li>• <code>LL_ADC_Enable ()</code></li> <li>• <code>LL_ADC_StartCalibration();</code></li> <li>• <code>LL_ADC_IsCalibrationOnGoing;</code></li> <li>• <code>LL_RCC_HSI_Enable ()</code></li> <li>• <code>LL_RCC_HSI_IsReady()</code></li> </ul>

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

**Table 23: Peripheral configuration management**

Name	Examples
<code>LL_PPP_{CATEGORY}_Set{ or Get}ConfigItem</code>	<code>LL_USART_SetBaudRate (USART2, 16000000,</code> <code>LL_USART_OVERSAMPLING_16, 9600)</code>

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

**Table 24: Peripheral register management**

Name
<code>LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)</code>
<code>LL_PPP_ReadReg(__INSTANCE__, __REG__)</code>
<code>LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel}, {uint32_t Proprietary})</code>



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

## 4 HAL and LL cohabitation

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 `stm32l0xx_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 STM32CubeL0 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 drivers 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 `stm32l0` 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 are 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\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_MspDelInit\(\)\*\*](#)
- [\*\*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
- Configure low power mode behavior when the MCU is in Debug mode
- Manage the VREFINT feature (activation, lock, output selection)

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_DBGMCU_DBG_EnableLowPowerConfig()`
- `HAL_DBGMCU_DBG_DisableLowPowerConfig()`
- `HAL_SYSCFG_GetBootMode()`
- `HAL_SYSCFG_VREFINT_OutputSelect()`
- `HAL_SYSCFG_Enable_Lock_VREFINT()`
- `HAL_SYSCFG_Disable_Lock_VREFINT()`

## 5.1.4 Detailed description of functions

### `HAL_Init`

Function Name	<code>HAL_StatusTypeDef HAL_Init (void )</code>
Function Description	This function configures the Flash prefetch, Flash preread and Buffer cache, Configures time base source, NVIC and Low level hardware.
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function is called at the beginning of program after reset and before the clock configuration</li> <li>• The time base configuration is based on MSI clock when exiting from Reset. Once done, time base tick start incrementing. In the default implementation,Systick is used as source of time base. the tick variable is incremented each 1ms in its ISR.</li> </ul>

### `HAL_DeInit`

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

**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.

- **TickPriority:** Tick interrupt priority.
- **HAL:** status
- 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 accurate delay (in ms) based on a variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.</li> <li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li> </ul>

### HAL\_GetTick

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

### HAL\_SuspendTick

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

### HAL\_ResumeTick

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

### HAL\_GetHalVersion

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

**HAL\_GetREVID**

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

**HAL\_GetDEVID**

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

**HAL\_DBGMCU\_EnableDBGSleepMode**

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

**HAL\_DBGMCU\_DisableDBGSleepMode**

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

**HAL\_DBGMCU\_EnableDBGStopMode**

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

**HAL\_DBGMCU\_DisableDBGStopMode**

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

**HAL\_DBGMCU\_EnableDBGStandbyMode**

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

**HAL\_DBGMCU\_DisableDBGStandbyMode**

Function Name	<b>void HAL_DBGMCU_DisableDBGStandbyMode (void )</b>
Function Description	Disables the Debug Module during STANDBY mode.

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

### **HAL\_DBGMCU\_DBG\_EnableLowPowerConfig**

Function Name	<b>void HAL_DBGMCU_DBG_EnableLowPowerConfig (uint32_t Periph)</b>
Function Description	Enable low power mode behavior when the MCU is in Debug mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> specifies the low power mode. This parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>– DBGMCU_SLEEP: Keep debugger connection during SLEEP mode</li> <li>– DBGMCU_STOP: Keep debugger connection during STOP mode</li> <li>– DBGMCU_STANDBY: Keep debugger connection during STANDBY mode</li> </ul> </li> </ul>

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

### **HAL\_DBGMCU\_DBG\_DisableLowPowerConfig**

Function Name	<b>void HAL_DBGMCU_DBG_DisableLowPowerConfig (uint32_t Periph)</b>
Function Description	Disable low power mode behavior when the MCU is in Debug mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> specifies the low power mode. This parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>– DBGMCU_SLEEP: Keep debugger connection during SLEEP mode</li> <li>– DBGMCU_STOP: Keep debugger connection during STOP mode</li> <li>– DBGMCU_STANDBY: Keep debugger connection during STANDBY mode</li> </ul> </li> </ul>

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

### **HAL\_SYSCFG\_GetBootMode**

Function Name	<b>uint32_t HAL_SYSCFG_GetBootMode (void )</b>
Function Description	Returns the boot mode as configured by user.
Return values	<ul style="list-style-type: none"> <li>• <b>The:</b> boot mode as configured by user. The returned value can be one of the following values: <ul style="list-style-type: none"> <li>– 0x00000000 : Boot is configured in Main Flash memory</li> <li>– 0x00000100 : Boot is configured in System Flash memory</li> <li>– 0x00000300 : Boot is configured in Embedded SRAM memory</li> </ul> </li> </ul>

### **HAL\_SYSCFG\_Enable\_Lock\_VREFINT**

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

---

Function Description	Lock the SYSCFG VREF register values.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### **HAL\_SYSCFG\_Disable\_Lock\_VREFINT**

Function Name	<b>void HAL_SYSCFG_Disable_Lock_VREFINT (void )</b>
Function Description	Unlock the overall SYSCFG VREF register values.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### **HAL\_SYSCFG\_VREFINT\_OutputSelect**

Function Name	<b>void HAL_SYSCFG_VREFINT_OutputSelect (uint32_t SYSCFG_Vrefint_OUTPUT)</b>
Function Description	Selects the output of internal reference voltage (VREFINT).
Parameters	<ul style="list-style-type: none"><li>• <b>SYSCFG_Vrefint_OUTPUT:</b> new state of the Vrefint output. This parameter can be one of the following values:<ul style="list-style-type: none"><li>- SYSCFG_VREFINT_OUT_NONE</li><li>- SYSCFG_VREFINT_OUT_PB0</li><li>- SYSCFG_VREFINT_OUT_PB1</li><li>- SYSCFG_VREFINT_OUT_PB0_PB1</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

## **5.2 HAL Firmware driver defines**

### **5.2.1 HAL**

#### ***DBGMCU Low Power Configuration***

DBGMCU\_SLEEP

DBGMCU\_STOP

DBGMCU\_STANDBY

IS\_DBGMCU\_PERIPH

#### ***HAL Exported Macros***

<code>_HAL_DBGMCU_FREEZE_TIM2</code>	TIM2 Peripherals Debug mode
<code>_HAL_DBGMCU_UNFREEZE_TIM2</code>	
<code>_HAL_DBGMCU_FREEZE_TIM3</code>	
<code>_HAL_DBGMCU_UNFREEZE_TIM3</code>	
<code>_HAL_DBGMCU_FREEZE_TIM6</code>	
<code>_HAL_DBGMCU_UNFREEZE_TIM6</code>	
<code>_HAL_DBGMCU_FREEZE_TIM7</code>	
<code>_HAL_DBGMCU_UNFREEZE_TIM7</code>	
<code>_HAL_DBGMCU_FREEZE_RTC</code>	
<code>_HAL_DBGMCU_UNFREEZE_RTC</code>	

---

```

__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_DBGMCU
__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT_D
__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT_DBGMCU
__HAL_DBGMCU_FREEZE_I2C3_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT_DBGMCU
__HAL_DBGMCU_FREEZE_LPTIMER
__HAL_DBGMCU_UNFREEZE_LPTIMER
__HAL_DBGMCU_FREEZE_TIM22
__HAL_DBGMCU_UNFREEZE_TIM22
__HAL_DBGMCU_FREEZE_TIM21
__HAL_DBGMCU_UNFREEZE_TIM21
__HAL_SYSCFG_REMAPMEMORY_FLASH
__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH
__HAL_SYSCFG_REMAPMEMORY_SRAM
__HAL_SYSCFG_DBG_LP_CONFIG

```

**Description:**

- Configuration of the DBG Low Power mode.

**Parameters:**

- DBGLPMODE: bit field to indicate in which Low Power mode DBG is still active. This parameter can be a value of
  - DBGMCU\_SLEEP
  - DBGMCU\_STOP
  - DBGMCU\_STANDBY

\_\_HAL\_SYSCFG\_GET\_BOOT\_MODE

**Description:**

- Returns the boot mode as configured by user.

**Return value:**

- The boot mode as configured by user. The returned can be a value of :

- SYSCFG\_BOOT\_MAINFLASH
- SYSCFG\_BOOT\_SYSTEMFLASH
- SYSCFG\_BOOT\_SRAM

### \_\_HAL\_SYSCFG\_GET\_FLAG

**Description:**

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

**Parameters:**

- \_\_FLAG\_\_: specifies the flag to check. The only parameter supported is SYSCFG\_FLAG\_VREFINT\_READ

**Return value:**

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

### \_\_HAL\_SYSCFG\_FASTMODEPLUS\_ENABLE

**Description:**

- Fast mode Plus driving capability enable macro.

**Parameters:**

- \_\_FASTMODEPLUS\_\_: This parameter can be a value of :
  - SYSCFG\_FASTMODEPLUS\_PB6
  - SYSCFG\_FASTMODEPLUS\_PB7
  - SYSCFG\_FASTMODEPLUS\_PB8
  - SYSCFG\_FASTMODEPLUS\_PB9

### \_\_HAL\_SYSCFG\_FASTMODEPLUS\_DISABLE

**Description:**

- Fast mode Plus driving capability disable macro.

**Parameters:**

- \_\_FASTMODEPLUS\_\_: This parameter can be a value of :
  - SYSCFG\_FASTMODEPLUS\_PB6
  - SYSCFG\_FASTMODEPLUS\_PB7
  - SYSCFG\_FASTMODEPLUS\_PB8
  - SYSCFG\_FASTMODEPLUS\_PB9

***HAL Version***

\_\_STM32L0xx\_HAL\_VERSION\_MAIN [31:24] main version  
\_\_STM32L0xx\_HAL\_VERSION\_SUB1 [23:16] sub1 version  
\_\_STM32L0xx\_HAL\_VERSION\_SUB2 [15:8] sub2 version  
\_\_STM32L0xx\_HAL\_VERSION\_RC [7:0] release candidate  
\_\_STM32L0xx\_HAL\_VERSION  
IDCODE\_DEVID\_MASK

**Boot Mode**

SYSCFG\_BOOT\_MAINFLASH  
SYSCFG\_BOOT\_SYSTEMFLASH  
SYSCFG\_BOOT\_SRAM

**Fast Mode Plus on GPIO**

SYSCFG\_FASTMODEPLUS\_PB6  
SYSCFG\_FASTMODEPLUS\_PB7  
SYSCFG\_FASTMODEPLUS\_PB8  
SYSCFG\_FASTMODEPLUS\_PB9  
IS\_SYSCFG\_FASTMODEPLUS

**SYSCFG Flags Definition**

SYSCFG\_FLAG\_VREFINT\_READY  
IS\_SYSCFG\_FLAG

**SYSCFG LCD External Capacitors**

SYSCFG_LCD_EXT_CAPA	Connection of internal Vlcd rail to external capacitors
SYSCFG_VLCD_PB2_EXT_CAPA_ON	Connection on PB2
SYSCFG_VLCD_PB12_EXT_CAPA_ON	Connection on PB12
SYSCFG_VLCD_PE11_EXT_CAPA_ON	Connection on PB0
SYSCFG_VLCD_PB0_EXT_CAPA_ON	Connection on PE11
SYSCFG_VLCD_PE12_EXT_CAPA_ON	Connection on PE12

**SYSCFG VREFINT Out Selection**

SYSCFG\_VREFINT\_OUT\_NONE  
SYSCFG\_VREFINT\_OUT\_PB0  
SYSCFG\_VREFINT\_OUT\_PB1  
SYSCFG\_VREFINT\_OUT\_PB0\_PB1  
IS\_SYSCFG\_VREFINT\_OUT\_SELECT

## 6 HAL ADC Generic Driver

### 6.1 ADC Firmware driver registers structures

#### 6.1.1 ADC\_OversamplingTypeDef

##### Data Fields

- *uint32\_t Ratio*
- *uint32\_t RightBitShift*
- *uint32\_t TriggeredMode*

##### Field Documentation

- *uint32\_t ADC\_OversamplingTypeDef::Ratio*  
Configures the oversampling ratio. This parameter can be a value of [\*ADC\\_Oversampling\\_Ratio\*](#)
- *uint32\_t ADC\_OversamplingTypeDef::RightBitShift*  
Configures the division coefficient for the Oversampler. This parameter can be a value of [\*ADC\\_Right\\_Bit\\_Shift\*](#)
- *uint32\_t ADC\_OversamplingTypeDef::TriggeredMode*  
Selects the regular triggered oversampling mode This parameter can be a value of [\*ADC\\_Triggered\\_Oversampling\\_Mode\*](#)

#### 6.1.2 ADC\_InitTypeDef

##### Data Fields

- *uint32\_t OversamplingMode*
- *ADC\_OversamplingTypeDef Oversample*
- *uint32\_t ClockPrescaler*
- *uint32\_t Resolution*
- *uint32\_t SamplingTime*
- *uint32\_t ScanConvMode*
- *uint32\_t DataAlign*
- *uint32\_t ContinuousConvMode*
- *uint32\_t DiscontinuousConvMode*
- *uint32\_t ExternalTrigConv*
- *uint32\_t ExternalTrigConvEdge*
- *uint32\_t DMAContinuousRequests*
- *uint32\_t EOCSelection*
- *uint32\_t Overrun*
- *uint32\_t LowPowerAutoWait*
- *uint32\_t LowPowerFrequencyMode*
- *uint32\_t LowPowerAutoPowerOff*

##### Field Documentation

- ***uint32\_t ADC\_InitTypeDef::OversamplingMode***  
Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***ADC\_OversamplingTypeDef ADC\_InitTypeDef::Oversample***  
Specifies the Oversampling parameters. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::ClockPrescaler***  
Selects the ADC clock frequency. This parameter can be a value of ***ADC\_ClockPrescaler***. Note: This parameter can be modified only if ADC is disabled.  
Note: In case of Synchronous clock mode divided by 1, this configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must be 50% duty cycle). Refer to reference manual for details
- ***uint32\_t ADC\_InitTypeDef::Resolution***  
Configures the ADC resolution mode. This parameter can be a value of ***ADC\_Resolution***. Note: This parameter can be modified only if ADC is disabled.
- ***uint32\_t ADC\_InitTypeDef::SamplingTime***  
The sample time value to be set for all channels. This parameter can be a value of ***ADC\_sampling\_times***. Note: This parameter can be modified only if there is no conversion ongoing.
- ***uint32\_t ADC\_InitTypeDef::ScanConvMode***  
The scan sequence direction. If several channels are set: Conversions are performed in sequence mode (ranks defined by each channel number: channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). This parameter can be a value of ***ADC\_Scan\_mode***. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::DataAlign***  
Specifies whether the ADC data alignment is left or right. This parameter can be a value of ***ADC\_data\_align***. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::ContinuousConvMode***  
Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::DiscontinuousConvMode***  
Specifies whether the conversion is performed in Complete-sequence/Discontinuous-sequence. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::ExternalTrigConv***  
Select the external event used to trigger the start of conversion. If set to ADC\_SOFTWARE\_START, external triggers are disabled. This parameter can be a value of ***ADC\_External\_trigger\_Source***. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::ExternalTrigConvEdge***  
Select the external trigger edge and enable the trigger. If trigger is set to ADC\_SOFTWARE\_START, this parameter is discarded. This parameter can be a value of ***ADC-Regular\_External\_Trigger\_Source\_Edge***. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::DMAContinuousRequests***  
Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer

- max pointer is reached. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32\_t ADC\_InitTypeDef::EOCSelection***  
Specifies what EOC (End Of Conversion) flag is used for conversion polling and interruption: end of single channel conversion or end of channels conversions sequence. This parameter can be a value of [\*\*ADC\\_EOCSelection\*\*](#)
  - ***uint32\_t ADC\_InitTypeDef::Overrun***  
Select the behaviour in case of overrun: data preserved or overwritten. This parameter has an effect on regular channels only, including in DMA mode. This parameter can be a value of [\*\*ADC\\_Overrun\*\*](#). Note: This parameter can be modified only if there is no conversion is ongoing.
  - ***uint32\_t ADC\_InitTypeDef::LowPowerAutoWait***  
Specifies the usage of dynamic low power Auto Delay: new conversion start only when the previous conversion (for regular channel) is completed. This avoids risk of overrun for low frequency application. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
  - ***uint32\_t ADC\_InitTypeDef::LowPowerFrequencyMode***  
When selecting an analog ADC clock frequency lower than 2.8MHz, it is mandatory to first enable the Low Frequency Mode. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
  - ***uint32\_t ADC\_InitTypeDef::LowPowerAutoPowerOff***  
When setting the AutoOff feature, the ADC is always powered off when not converting and automatically wakes-up when a conversion is started (by software or hardware trigger). This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.

### 6.1.3 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

## 6.1.4 ADC\_ChannelConfTypeDef

### Data Fields

- *`uint32_t Channel`*
- *`uint32_t Rank`*

### Field Documentation

- *`uint32_t ADC_ChannelConfTypeDef::Channel`*  
the ADC channel to configure This parameter can be a value of [ADC\\_channels](#)
- *`uint32_t ADC_ChannelConfTypeDef::Rank`*  
Add or remove the channel from ADC regular group sequencer. On STM32L0 devices, number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Despite the channel rank is fixed, this parameter allow an additional possibility: to remove the selected rank (selected channel) from sequencer. This parameter can be a value of [ADC\\_rank](#)

## 6.1.5 ADC\_AnalogWDGConfTypeDef

### Data Fields

- *`uint32_t WatchdogMode`*
- *`uint32_t Channel`*
- *`uint32_t ITMode`*
- *`uint32_t HighThreshold`*
- *`uint32_t LowThreshold`*

### Field Documentation

- *`uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode`*  
Configures the ADC analog watchdog mode: single/all channels. 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. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

- ***uint32\_t ADC\_AnalogWDGConfTypeDef::LowThreshold***  
Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

## 6.2 ADC Firmware driver API description

### 6.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- A built-in hardware oversampler can handle multiple conversions and average them into a single data with increased data width, up to 16-bit.
- Interrupt generation at the end of regular 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 (common for all channels)
- ADC conversion of regular group.
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- 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 STM32L0, ADC clock frequency max is 16MHz (refer to device datasheet). Therefore, ADC clock prescaler must be configured in function of ADC clock source frequency to remain below this maximum frequency.
  - Two clock settings are mandatory:
    - ADC clock (core clock, also possibly conversion clock).
    - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 16MHz. If asynchronous clock is selected, parameter "HSIState" must be set either: - to "...HSIState = RCC\_HSI\_ON" to maintain the HSI16 oscillator always enabled: can be used to supply the main system clock.
    - Example: Into HAL\_ADC\_MspInit() (recommended code location) or with other device clock parameters configuration:
    - \_\_HAL\_RCC\_ADC1\_CLK\_ENABLE(); (mandatory) HSI16 enable : (optional: if asynchronous clock selected)

- RCC\_OsclInitTypeDef RCC\_OsclInitStructure;
- RCC\_OsclInitStructure.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;
- RCC\_OsclInitStructure.HSI16CalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;
- RCC\_OsclInitStructure.HSISState = RCC\_HSI\_ON;
- RCC\_OsclInitStructure.PLL... (optional if used for system clock)
- HAL\_RCC\_OscConfig(&RCC\_OsclInitStructure);
- ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL\_ADC\_Init().
- 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, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, oversampler, continuous mode, ...) 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 analog watchdog parameters (channels monitored, thresholds, ...) using function HAL\_ADC\_AnalogWDGConfig().
4. When device is in mode low-power (low-power run, low-power sleep or stop mode), function "HAL\_ADCEx\_EnableVREFINT()" must be called before function HAL\_ADC\_Init(). In case of internal temperature sensor to be measured: function "HAL\_ADCEx\_EnableVREFINTTempSensor()" must be called similarly

### 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()
    - Retrieve conversion results using function HAL\_ADC\_GetValue()

- 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)
  - Retrieve conversion results using function HAL\_ADC\_GetValue()
  - 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()



Callback functions must be implemented in user program:

- HAL\_ADC\_ErrorCallback()
- HAL\_ADC\_LevelOutOfWindowCallback() (callback of analog watchdog)
- HAL\_ADC\_ConvCpltCallback()
- HAL\_ADC\_ConvHalfCpltCallback()

## 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\_MspDelinit() (recommended code location) or with other device clock parameters configuration:
    - RCC\_OscInitStructure.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;
    - RCC\_OscInitStructure.HSISState = RCC\_HSI\_OFF; (if not used for system clock)
    - HAL\_RCC\_OscConfig(&RCC\_OscInitStructure);
2. ADC pins configuration
  - Disable the clock for the ADC GPIOs using macro \_\_HAL\_RCC\_GPIOx\_CLK\_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
  - Disable the NVIC for ADC using function HAL\_NVIC\_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 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\(\)\*](#)

### 6.2.4 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\(\)\*](#)

### 6.2.5 ADC Peripheral State functions

This subsection provides functions allowing to

- Check the ADC state.
- handle ADC interrupt request.

This section contains the following APIs:

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

## 6.2.6 Detailed description of functions

### HAL\_ADC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)</b>
Function Description	Initializes the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• As prerequisite, ADC clock must be configured at RCC top level depending on both possible clock sources: APB clock or HSI clock. See commented example code below that can be copied and uncommented into HAL_ADC_MspInit().</li> <li>• 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_Delnit() 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".</li> <li>• 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".</li> <li>• When device is in mode low-power (low-power run, low-power sleep or stop mode), function "HAL_ADCEx_EnableVREFINT()" must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_Delnit() must be called first). In case of internal temperature sensor to be measured: function "HAL_ADCEx_EnableVREFINTTempSensor()" must be called similarly.</li> </ul>

### HAL\_ADC\_Delnit

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Delnit (ADC_HandleTypeDef * hadc)</b>
Function Description	Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common</li> </ul>

group is still running.

### **HAL\_ADC\_MspInit**

Function Name	<b>void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)</b>
Function Description	Initializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_ADC\_MspDeInit**

Function Name	<b>void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)</b>
Function Description	Deinitializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_ADC\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)</b>
Function Description	Enables ADC, starts conversion of regular group.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_ADC\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)</b>
Function Description	Stop ADC conversion of regular group, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status.</li> </ul>

### **HAL\_ADC\_PollForConversion**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)</b>
Function Description	Wait for regular group conversion to be completed.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> <li>• <b>Timeout:</b> Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue().</li> <li>• This function cannot be used in a particular setup: ADC</li> </ul>

configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC\_EOC\_SINGLE\_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC\_EOC\_SEQ\_CONV).

### **HAL\_ADC\_PollForEvent**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)</b>
Function Description	Poll for conversion event.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> <li>• <b>EventType:</b> the ADC event type. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– ADC_AWD_EVENT: ADC Analog watchdog event</li> <li>– ADC_OVR_EVENT: ADC Overrun event</li> </ul> </li> <li>• <b>Timeout:</b> Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_ADC\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)</b>
Function Description	Enables ADC, starts conversion of regular group with interruption.

### **HAL\_ADC\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)</b>
Function Description	Stop ADC conversion of regular group, disable interruption of end-of-conversion, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status.</li> </ul>

### **HAL\_ADC\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)</b>
Function Description	Enables ADC, starts conversion of regular group and transfers result through DMA.

### **HAL\_ADC\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)</b>
---------------	--

---

Function Description	Stop ADC conversion of regular group, disable ADC DMA transfer, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status.</li> </ul>

### **HAL\_ADC\_GetValue**

Function Name	<b>uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)</b>
Function Description	Get ADC regular group conversion result.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Converted:</b> value</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Reading DR register automatically clears EOC (end of conversion of regular group) flag.</li> <li>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(&amp;hadc, ADC_FLAG_EOS).</li> </ul>

### **HAL\_ADC\_IRQHandler**

Function Name	<b>void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)</b>
Function Description	Handles ADC interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### **HAL\_ADC\_ConvCpltCallback**

Function Name	<b>void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	Conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### **HAL\_ADC\_ConvHalfCpltCallback**

Function Name	<b>void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	Conversion DMA half-transfer callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

**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:  
VrefInt/Vlcd(STM32L0x3xx only)/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\_vlcd (STM32L0x3xx only), TS\_temp (values rough order: 5us to 17us). These internal paths can be disabled using function HAL\_ADC\_Delnit().  
•    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 resetting 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	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> <li><b>AnalogWDGConfig:</b> Structure of ADC analog watchdog configuration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Possibility to update parameters on the fly: This function initializes the selected analog watchdog, following calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".</li> </ul>

### **HAL\_ADC\_GetState**

Function Name	<b>uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)</b>
Function Description	return the ADC state
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> state</li> </ul>

### **HAL\_ADC\_GetError**

Function Name	<b>uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)</b>
Function Description	Return the ADC error code.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>ADC:</b> Error Code</li> </ul>

## **6.3 ADC Firmware driver defines**

### **6.3.1 ADC**

#### ***ADC Analog Watchdog Mode***

ADC\_ANALOGWATCHDOG\_NONE  
 ADC\_ANALOGWATCHDOG\_SINGLE\_REG  
 ADC\_ANALOGWATCHDOG\_ALL\_REG

#### ***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\_18  
ADC\_CHANNEL\_VLCD  
ADC\_CHANNEL\_VREFINT  
ADC\_CHANNEL\_TEMPSENSOR

***ADC Channel Masks***

ADC\_CHANNEL\_MASK  
ADC\_CHANNEL\_AWD\_MASK

***ADC Clock Prescaler***

ADC_CLOCK_ASYNC_DIV1	ADC Asynchronous clock mode divided by 1
ADC_CLOCK_ASYNC_DIV2	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV4	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV6	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV8	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV10	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV12	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV16	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV32	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV64	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV128	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV256	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_SYNC_PCLK_DIV1	Synchronous clock mode divided by 1 This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle)
ADC_CLOCK_SYNC_PCLK_DIV2	Synchronous clock mode divided by 2
ADC_CLOCK_SYNC_PCLK_DIV4	Synchronous clock mode divided by 4

***ADC Conversion Group***

ADC\_REGULAR\_GROUP

***ADC Data Align***

ADC\_DATAALIGN\_RIGHT

ADC\_DATAALIGN\_LEFT

***ADC EOC Selection***

ADC\_EOC\_SINGLE\_CONV

ADC\_EOC\_SEQ\_CONV

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

***ADC Error Code***

HAL\_ADC\_ERROR\_NONE No error

HAL\_ADC\_ERROR\_INTERNAL ADC IP internal error: if problem of clocking, enable/disable, erroneous state

HAL\_ADC\_ERROR\_OVR OVR error

HAL\_ADC\_ERROR\_DMA DMA transfer error

***ADC Event***

ADC\_AWD\_EVENT

ADC\_OVR\_EVENT

***ADC Exported Macro***

`_HAL_ADC_RESET_HANDLE_STATE`

**Description:**

- Reset ADC handle state.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None

**Description:**

- Enable the ADC peripheral.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None

`_HAL_ADC_ENABLE`

ADC\_ENABLING\_CONDITIONS

**Description:**

- Verification of hardware constraints before ADC can be enabled.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- SET: (ADC can be enabled) or  
RESET (ADC cannot be enabled)

`__HAL_ADC_DISABLE`

**Description:**

- Disable the ADC peripheral.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

`ADC_DISABLE_CONDITIONS`

**Description:**

- Verification of hardware constraints before ADC can be disabled.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- SET: (ADC can be disabled) or  
RESET (ADC cannot be disabled)

`ADC_IS_ENABLE`

**Description:**

- Verification of ADC state: enabled or disabled.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

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

`ADC_GET_RESOLUTION`

**Description:**

- Returns resolution bits in CFGR register: RES[1:0].

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

`ADC_IS_SOFTWARE_START_REGULAR`

**Description:**

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

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

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

`ADC_IS_CONVERSION_ONGOING_REGULAR`

**Description:**

- Check if no conversion on going on regular group.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- SET: (conversion is on going) or RESET (no conversion is on going)

`ADC_CONTINUOUS`

**Description:**

- Enable ADC continuous conversion mode.

**Parameters:**

- `_CONTINUOUS_MODE_`: Continuous mode.

**Return value:**

- None

`ADC_SCANDIR`

**Description:**

- Enable ADC scan mode to convert multiple ranks with sequencer.

**Parameters:**

- `_SCAN_MODE_`: Scan conversion mode.

**Return value:**

- None

`_HAL_ADC_CFGR1_DISCONTINUOUS_NUM`

**Description:**

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

**Parameters:**

- `_NBR_DISCONTINUOUS_CONV_`: Number of discontinuous conversions.

**Return value:**

- None

`ADC_DMACONTREQ`

**Description:**

- Enable the ADC DMA continuous request.

**Parameters:**

- `_DMAContReq_MODE_`: DMA continuous request mode.

**Return value:**

- None

`__HAL_ADC_CFGR1_AutoDelay`

**Description:**

- Enable the ADC Auto Delay.

**Parameters:**

- `_AutoDelay_`: Auto delay bit enable or disable.

**Return value:**

- None

`__HAL_ADC_CFGR1_AUTOFF`

**Description:**

- Enable the ADC LowPowerAutoPowerOff.

**Parameters:**

- `_AUTOFF_`: AutoOff bit enable or disable.

**Return value:**

- None

`ADC_TRX_HIGHTHRESHOLD`

**Description:**

- Configure the analog watchdog high threshold into registers TR1, TR2 or TR3.

**Parameters:**

- `_Threshold_`: Threshold value

**Return value:**

- None

`__HAL_ADC_CCR_LOWFREQUENCY`

**Description:**

- Enable the ADC Low Frequency mode.

**Parameters:**

- `_LOW_FREQUENCY_MODE_`: Low Frequency mode.

**Return value:**

- None

`ADC_OFFSET_SHIFT_RESOLUTION`

**Description:**

- Shift the offset in function of the selected ADC resolution.

`ADC_AWD1THRESHOLD_SHIFT_RESOLUTION`

**Parameters:**

- `_HANDLE_`: ADC handle.
- `_Offset_`: Value to be shifted

**Return value:**

- None

**Description:**

- Shift the AWD1 threshold in function of the selected ADC resolution.

`__HAL_ADC_Value_Shift_left`

**Parameters:**

- `_HANDLE_`: ADC handle.
- `_Threshold_`: Value to be shifted

**Return value:**

- None

**Description:**

- Shift the value on the left, less significant are set to 0.

`__HAL_ADC_ENABLE_IT`

**Parameters:**

- `_Value_`: Value to be shifted
- `_Shift_`: Number of shift to be done

**Return value:**

- None

**Description:**

- Enable the ADC end of conversion interrupt.

`__HAL_ADC_DISABLE_IT`

**Parameters:**

- `_HANDLE_`: ADC handle.
- `_INTERRUPT_`: ADC Interrupt.

**Return value:**

- None

**Description:**

- Disable the ADC end of conversion interrupt.

`__HAL_ADC_GET_IT_SOURCE`

**Parameters:**

- `_HANDLE_`: ADC handle.
- `_INTERRUPT_`: ADC interrupt.

**Return value:**

- None

**Description:**

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

**Parameters:**

- `__HANDLE__`: ADC handle
- `__INTERRUPT__`: ADC interrupt source to check
  - ...
  - ...

**Return value:**

- State: of interruption (TRUE or FALSE)

`_HAL_ADC_CLEAR_FLAG`

**Description:**

- Clear the ADC's pending flags.

**Parameters:**

- `__HANDLE__`: ADC handle.
- `__FLAG__`: ADC flag.

**Return value:**

- None

`_HAL_ADC_GET_FLAG`

**Description:**

- Get the selected ADC's flag status.

**Parameters:**

- `__HANDLE__`: ADC handle.
- `__FLAG__`: ADC flag.

**Return value:**

- None

`ADC_STATE_CLR_SET`

**Description:**

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

**Return value:**

- None

**Notes:**

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

`ADC_CLEAR_ERRORCODE`

**Description:**

- Clear ADC error code (set it to

---

error code: "no error")

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

**Description:**

- Configuration of ADC clock & prescaler: clock source PCLK or Asynchronous with selectable prescaler.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

`IS_ADC_CLOCKPRESCALER`

`IS_ADC_RESOLUTION`

`IS_ADC_RESOLUTION_8_6_BITS`

`IS_ADC_DATA_ALIGN`

`IS_ADC_EXTTRIG_EDGE`

`IS_ADC_EOC_SELECTION`

`IS_ADC_OVERRUN`

`IS_ADC_RANK`

`IS_ADC_CHANNEL`

`IS_ADC_SAMPLE_TIME`

`IS_ADC_SCAN_MODE`

`IS_ADC_OVERSAMPLING_RATIO`

`IS_ADC_RIGHT_BIT_SHIFT`

`IS_ADC_TRIGGERED_OVERSAMPLING_MODE`  
E

`IS_ADC_ANALOG_WATCHDOG_MODE`

`IS_ADC_CONVERSION_GROUP`

`IS_ADC_EVENT_TYPE`

***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	Overrun occurrence
HAL_ADC_STATE_REG_EOSMP	Not available on STM32F0 device: End Of Sampling flag raised
HAL_ADC_STATE_INJ_BUSY	Not available on STM32F0 device: 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	Not available on STM32F0 device: Conversion data available on group injected
HAL_ADC_STATE_INJ_JQOVF	Not available on STM32F0 device: Not available on STM32F0 device: Injected queue overflow occurrence
HAL_ADC_STATE_AWD1	Out-of-window occurrence of analog watchdog 1
HAL_ADC_STATE_AWD2	Not available on STM32F0 device: Out-of-window occurrence of analog watchdog 2
HAL_ADC_STATE_AWD3	Not available on STM32F0 device: Out-of-window occurrence of analog watchdog 3
HAL_ADC_STATE_MULTIMODE_SLAVE	Not available on STM32F0 device: ADC in multimode slave state, controlled by another ADC master (

#### ***ADC External Trigger Source***

ADC\_EXTERNALTRIGCONV\_T6\_TRGO  
 ADC\_EXTERNALTRIGCONV\_T21\_CC2  
 ADC\_EXTERNALTRIGCONV\_T2\_TRGO  
 ADC\_EXTERNALTRIGCONV\_T2\_CC4  
 ADC\_EXTERNALTRIGCONV\_T22\_TRGO  
 ADC\_EXTERNALTRIGCONV\_T3\_TRGO  
 ADC\_EXTERNALTRIGCONV\_EXT\_IT11  
 ADC\_SOFTWARE\_START  
 ADC\_EXTERNALTRIGCONV\_T21\_TRGO  
 ADC\_EXTERNALTRIGCONV\_T2\_CC3  
 IS\_ADC\_EXTRIG

***ADC Flags Definition***

ADC_FLAG_RDY	ADC Ready (ADRDY) flag
ADC_FLAG_EOSMP	ADC End of Sampling flag
ADC_FLAG_EOC	ADC End of Regular Conversion flag
ADC_FLAG_EOS	ADC End of Regular sequence of Conversions flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_AWD	ADC Analog watchdog flag
ADC_FLAG_EOCAL	ADC Enf Of Calibration flag
ADC_FLAG_ALL	

***ADC Interrupts Definition***

ADC_IT_RDY	ADC Ready (ADRDY) interrupt source
ADC_IT_EOSMP	ADC End of Sampling interrupt source
ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_EOS	ADC End of Regular sequence of Conversions interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_AWD	ADC Analog watchdog 1 interrupt source
ADC_IT_EOCAL	ADC End of Calibration interrupt source

***ADC Overrun***

ADC_OVR_DATA_PRESERVED	
ADC_OVR_DATA_OVERWRITTEN	

***ADC Oversampling Ratio***

ADC_OVERSAMPLING_RATIO_2	ADC Oversampling ratio 2x
ADC_OVERSAMPLING_RATIO_4	ADC Oversampling ratio 4x
ADC_OVERSAMPLING_RATIO_8	ADC Oversampling ratio 8x
ADC_OVERSAMPLING_RATIO_16	ADC Oversampling ratio 16x
ADC_OVERSAMPLING_RATIO_32	ADC Oversampling ratio 32x
ADC_OVERSAMPLING_RATIO_64	ADC Oversampling ratio 64x
ADC_OVERSAMPLING_RATIO_128	ADC Oversampling ratio 128x
ADC_OVERSAMPLING_RATIO_256	ADC Oversampling ratio 256x

***ADC Range Verification***

IS\_ADC\_RANGE

***ADC rank***

ADC_RANK_CHANNEL_NUMBER	Enable the rank of the selected channels. Number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...)
ADC_RANK_NONE	Disable the selected rank (selected channel) from

**sequencer*****ADC External Trigger Source Edge for Regular Group***

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

***ADC Regular Nb Conversion Verification***

IS\_ADC\_REGULAR\_NB\_CONV

***ADC Resolution***

ADC\_RESOLUTION\_12B ADC 12-bit resolution  
ADC\_RESOLUTION\_10B ADC 10-bit resolution  
ADC\_RESOLUTION\_8B ADC 8-bit resolution  
ADC\_RESOLUTION\_6B ADC 6-bit resolution

***ADC Right Bit Shift***

ADC\_RIGHTBITSHIFT\_NONE ADC No bit shift for oversampling  
ADC\_RIGHTBITSHIFT\_1 ADC 1 bit shift for oversampling  
ADC\_RIGHTBITSHIFT\_2 ADC 2 bits shift for oversampling  
ADC\_RIGHTBITSHIFT\_3 ADC 3 bits shift for oversampling  
ADC\_RIGHTBITSHIFT\_4 ADC 4 bits shift for oversampling  
ADC\_RIGHTBITSHIFT\_5 ADC 5 bits shift for oversampling  
ADC\_RIGHTBITSHIFT\_6 ADC 6 bits shift for oversampling  
ADC\_RIGHTBITSHIFT\_7 ADC 7 bits shift for oversampling  
ADC\_RIGHTBITSHIFT\_8 ADC 8 bits shift for oversampling

***ADC Sampling Cycles***

ADC\_SAMPLETIME\_1CYCLE\_5 ADC sampling time 1.5 cycle  
ADC\_SAMPLETIME\_7CYCLES\_5 ADC sampling time 7.5 CYCLES  
ADC\_SAMPLETIME\_13CYCLES\_5 ADC sampling time 13.5 CYCLES  
ADC\_SAMPLETIME\_28CYCLES\_5 ADC sampling time 28.5 CYCLES  
ADC\_SAMPLETIME\_41CYCLES\_5 ADC sampling time 41.5 CYCLES  
ADC\_SAMPLETIME\_55CYCLES\_5 ADC sampling time 55.5 CYCLES  
ADC\_SAMPLETIME\_71CYCLES\_5 ADC sampling time 71.5 CYCLES  
ADC\_SAMPLETIME\_239CYCLES\_5 ADC sampling time 239.5 CYCLES

***ADC Scan mode***

ADC\_SCAN\_DIRECTION\_FORWARD Scan direction forward: from channel 0 to channel 18  
ADC\_SCAN\_DIRECTION\_BACKWARD Scan direction backward: from channel 18 to channel 0

ADC\_SCAN\_ENABLE

***ADC SYSCFG internal paths Flags Definition***

ADC\_FLAG\_SENSOR

ADC\_FLAG\_VREFINT

***ADC TimeOut Values***

ADC\_ENABLE\_TIMEOUT

ADC\_DISABLE\_TIMEOUT

ADC\_STOP\_CONVERSION\_TIMEOUT

ADC\_DELAY\_10US\_MIN\_CPU\_CYCLES

***ADC Triggered Oversampling Mode***

ADC\_TRIGGEREDMODE\_SINGLE\_TRIGGER    ADC No bit shift for oversampling

ADC\_TRIGGEREDMODE\_MULTI\_TRIGGER    ADC No bit shift for oversampling

## 7 HAL ADC Extension Driver

### 7.1 ADCEx Firmware driver API description

#### 7.1.1 ADC specific features

1. Self calibration.

#### 7.1.2 How to use this driver

1. Call HAL\_ADCEx\_Calibration\_Start() to start calibration
2. Read the calibration factor using HAL\_ADCEx\_Calibration\_GetValue()
3. User can set a his calibration factor using HAL\_ADCEx\_Calibration\_SetValue()

#### 7.1.3 Detailed description of functions

##### HAL\_ADCEx\_Calibration\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)</b>
Function Description	Start an automatic calibration.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc:</b> pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li><li>• <b>SingleDiff:</b> Selection of single-ended or differential input This parameter can be only of the following values:<ul style="list-style-type: none"><li>– ADC_SINGLE_ENDED: Channel in mode input single ended</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

##### HAL\_ADCEx\_Calibration\_GetValue

Function Name	<b>uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)</b>
Function Description	Get the calibration factor.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc:</b> ADC handle.</li><li>• <b>SingleDiff:</b> This parameter can be only:<ul style="list-style-type: none"><li>– ADC_SINGLE_ENDED: Channel in mode input single ended.</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>Calibration:</b> value.</li></ul>

##### HAL\_ADCEx\_Calibration\_SetValue

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)</b>
---------------	---

Function Description	Set the calibration factor to overwrite automatic conversion result.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> <li>• <b>SingleDiff:</b> This parameter can be only: <ul style="list-style-type: none"> <li>– ADC_SINGLE_ENDED: Channel in mode input single ended.</li> </ul> </li> <li>• <b>CalibrationFactor:</b> Calibration factor (coded on 7 bits maximum)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

### **HAL\_ADCEx\_EnableVREFINT**

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_EnableVREFINT (void )</b>
Function Description	Enables the buffer of Vrefint for the ADC, required when device is in mode low-power (low-power run, low-power sleep or stop mode) This function must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_Delnit() must be called first) For more details on procedure and buffer current consumption, refer to device reference manual.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This is functional only if the LOCK is not set.</li> </ul>

### **HAL\_ADCEx\_DisableVREFINT**

Function Name	<b>void HAL_ADCEx_DisableVREFINT (void )</b>
Function Description	Disables the Buffer Vrefint for the ADC.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This is functional only if the LOCK is not set.</li> </ul>

### **HAL\_ADCEx\_EnableVREFINTTempSensor**

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_EnableVREFINTTempSensor (void )</b>
Function Description	Enables the buffer of temperature sensor for the ADC, required when device is in mode low-power (low-power run, low-power sleep or stop mode) This function must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_Delnit() must be called first) For more details on procedure and buffer current consumption, refer to device reference manual.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This is functional only if the LOCK is not set.</li> </ul>

### **HAL\_ADCEx\_DisableVREFINTTempSensor**

Function Name	<b>void HAL_ADCEx_DisableVREFINTTempSensor (void )</b>
Function Description	Disables the VREFINT and Sensor for the ADC.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**Notes**

- This is functional only if the LOCK is not set.

## 7.2 ADCEx Firmware driver defines

### 7.2.1 ADCEx

#### *ADC Calibration Factor Length Verification*

`IS_ADC_CALFACT`

##### **Description:**

- Calibration factor length verification (7 bits maximum)

##### **Parameters:**

- `_Calibration_Factor_`: Calibration factor value

##### **Return value:**

- None

#### *ADC Single Ended*

`ADC_SINGLE_ENDED`

## 8 HAL COMP Generic Driver

### 8.1 COMP Firmware driver registers structures

#### 8.1.1 COMP\_InitTypeDef

##### Data Fields

- *uint32\_t WindowMode*
- *uint32\_t Mode*
- *uint32\_t NonInvertingInput*
- *uint32\_t InvertingInput*
- *uint32\_t OutputPol*
- *uint32\_t LPTIMConnection*
- *uint32\_t TriggerMode*

##### Field Documentation

- ***uint32\_t COMP\_InitTypeDef::WindowMode***  
Set window mode of a pair of comparators instances (2 consecutive instances odd and even COMP<x> and COMP<x+1>). Note: HAL COMP driver allows to set window mode from any COMP instance of the pair of COMP instances composing window mode. This parameter can be a value of [\*\*COMP\\_WindowMode\*\*](#)
- ***uint32\_t COMP\_InitTypeDef::Mode***  
Set comparator operating mode to adjust power and speed. Note: For the characteristics of comparator power modes (propagation delay and power consumption), refer to device datasheet. This parameter can be a value of [\*\*COMP\\_PowerMode\*\*](#)
- ***uint32\_t COMP\_InitTypeDef::NonInvertingInput***  
Set comparator input plus (non-inverting input). This parameter can be a value of [\*\*COMP\\_InputPlus\*\*](#)
- ***uint32\_t COMP\_InitTypeDef::InvertingInput***  
Set comparator input minus (inverting input). This parameter can be a value of [\*\*COMP\\_InputMinus\*\*](#)
- ***uint32\_t COMP\_InitTypeDef::OutputPol***  
Set comparator output polarity. This parameter can be a value of [\*\*COMP\\_OutputPolarity\*\*](#)
- ***uint32\_t COMP\_InitTypeDef::LPTIMConnection***  
Set comparator output connection to LPTIM peripheral. This parameter can be a value of [\*\*COMP\\_LPTIMConnection\*\*](#)
- ***uint32\_t COMP\_InitTypeDef::TriggerMode***  
Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of [\*\*COMP\\_EXTI\\_TriggerMode\*\*](#)

#### 8.1.2 COMP\_HandleTypeDef

##### Data Fields

- ***COMP\_TypeDef \* Instance***
- ***COMP\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_COMP\_StateTypeDef State***

#### Field Documentation

- ***COMP\_TypeDef\* COMP\_HandleTypeDef::Instance***  
Register base address
- ***COMP\_InitTypeDef COMP\_HandleTypeDef::Init***  
COMP required parameters
- ***HAL\_LockTypeDef COMP\_HandleTypeDef::Lock***  
Locking object
- ***\_\_IO HAL\_COMP\_StateTypeDef COMP\_HandleTypeDef::State***  
COMP communication state

## 8.2 COMP Firmware driver API description

### 8.2.1 COMP Peripheral features

The STM32L0xx device family integrates two analog comparators instances COMP1 and COMP2:

1. The COMP input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. The COMP output level is available using HAL\_COMP\_GetOutputLevel() and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. Pairs of comparators instances can be combined in window mode (2 consecutive instances odd and even COMP<x> and COMP<x+1>).
4. The comparators have interrupt capability through the EXTI controller with wake-up from sleep and stop modes:
  - COMP1 is internally connected to EXTI Line 21
  - COMP2 is internally connected to EXTI Line 22 From the corresponding IRQ handler, the right interrupt source can be retrieved using macro  
`__HAL_COMP_COMP1_EXTI_GET_FLAG()` and  
`__HAL_COMP_COMP2_EXTI_GET_FLAG()`.

### 8.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32L0xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the HAL\_COMP\_MspInit():
  - Configure the GPIO connected to comparator inputs plus and minus in analog mode using HAL\_GPIO\_Init().
  - If needed, configure the GPIO connected to comparator output in alternate function mode using HAL\_GPIO\_Init().
  - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL\_GPIO\_Init() function. After that enable the comparator interrupt vector using HAL\_NVIC\_EnableIRQ() function.
2. Configure the comparator using HAL\_COMP\_Init() function:
  - Select the input minus (inverting input)

- Select the input plus (non-inverting input)
  - Select the output polarity
  - Select the power mode
  - Select the window mode HAL\_COMP\_Init() calls internally \_\_HAL\_RCC\_SYSCFG\_CLK\_ENABLE() to enable internal control clock of the comparators. However, this is a legacy strategy. In future STM32 families, COMP clock enable must be implemented by user in "HAL\_COMP\_MspInit()". Therefore, for compatibility anticipation, it is recommended to implement \_\_HAL\_RCC\_SYSCFG\_CLK\_ENABLE() in "HAL\_COMP\_MspInit()".
3. Reconfiguration on-the-fly of comparator can be done by calling again function HAL\_COMP\_Init() with new input structure parameters values.
  4. Enable the comparator using HAL\_COMP\_Start() function.
  5. Use HAL\_COMP\_TriggerCallback() or HAL\_COMP\_GetOutputLevel() functions to manage comparator outputs (events and output level).
  6. Disable the comparator using HAL\_COMP\_Stop() function.
  7. De-initialize the comparator using HAL\_COMP\_DelInit() function.
  8. For safety purpose, comparator configuration can be locked using HAL\_COMP\_Lock() function. The only way to unlock the comparator is a device hardware reset.

### 8.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

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

### 8.2.4 IO operation functions

This section provides functions allowing to:

- Start a comparator instance.
- Stop a comparator instance.

This section contains the following APIs:

- [\*\*HAL\\_COMP\\_Start\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_Stop\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_IRQHandler\(\)\*\*](#)

### 8.2.5 Peripheral Control functions

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

This section contains the following APIs:

- [\*\*HAL\\_COMP\\_Lock\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_GetOutputLevel\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_TriggerCallback\(\)\*\*](#)

### 8.2.6 Peripheral State functions

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

This section contains the following APIs:

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

## 8.2.7 Detailed description of functions

### HAL\_COMP\_Init

Function Name	<b>HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)</b>
Function Description	Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.</li> <li>• When the LPTIM connection is enabled, the following pins LPTIM_IN1(PB5, PC0) and LPTIM_IN2(PB7, PC2) should not be configured in alternate function.</li> </ul>

### HAL\_COMP\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_COMP_DelInit (COMP_HandleTypeDef * hcomp)</b>
Function Description	Deinitialize the COMP peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.</li> </ul>

### HAL\_COMP\_MspInit

Function Name	<b>void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)</b>
Function Description	Initialize the COMP MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_COMP\_MspDelInit

Function Name	<b>void HAL_COMP_MspDelInit (COMP_HandleTypeDef * hcomp)</b>
Function Description	Deinitialize the COMP MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_COMP\_Start

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

**(COMP\_HandleTypeDef \* hcomp)**

Function Description	Start the comparator.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_COMP\_Stop****Function Name**      **HAL\_StatusTypeDef HAL\_COMP\_Stop  
(COMP\_HandleTypeDef \* hcomp)**

Function Description	Stop the comparator.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_COMP\_IRQHandler****Function Name**      **void HAL\_COMP\_IRQHandler (COMP\_HandleTypeDef \* hcomp)**

Function Description	Comparator IRQ handler.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_COMP\_Lock****Function Name**      **HAL\_StatusTypeDef HAL\_COMP\_Lock  
(COMP\_HandleTypeDef \* hcomp)**

Function Description	Lock the selected comparator configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• A system reset is required to unlock the comparator configuration.</li> <li>• Locking the comparator from reset state is possible if __HAL_RCC_SYSCFG_CLK_ENABLE() is being called before.</li> </ul>

**HAL\_COMP\_GetOutputLevel****Function Name**      **uint32\_t HAL\_COMP\_GetOutputLevel (COMP\_HandleTypeDef \* hcomp)**

Function Description	Return the output level (high or low) of the selected comparator.
----------------------	---

**HAL\_COMP\_TriggerCallback****Function Name**      **void HAL\_COMP\_TriggerCallback (COMP\_HandleTypeDef \* hcomp)**

Function Description	Comparator callback.
----------------------	----------------------

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

**HAL\_COMP\_GetState**

Function Name	<b>HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)</b>
Function Description	Return the COMP handle state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp:</b> COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

## 8.3 COMP Firmware driver defines

### 8.3.1 COMP

***COMP Exported Types***`COMP_STATE_BITFIELD_LOCK`***COMP EXTI Lines***

<code>COMP_EXTI_LINE_COMP1</code>	EXTI line 21 connected to COMP1 output
<code>COMP_EXTI_LINE_COMP2</code>	EXTI line 22 connected to COMP2 output
<code>COMP_EXTI_IT</code>	EXTI line event with interruption
<code>COMP_EXTI_EVENT</code>	EXTI line event only (without interruption)
<code>COMP_EXTI_RISING</code>	EXTI line event on rising edge
<code>COMP_EXTI_FALLING</code>	EXTI line event on falling edge

***COMP external interrupt line management***

<code>_HAL_COMP_COMP1_EXTI_ENABLE_RISING_ED GE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Enable the COMP1 EXTI line rising edge trigger.</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>
<code>_HAL_COMP_COMP1_EXTI_DISABLE_RISING_E DGE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Disable the COMP1 EXTI line rising edge trigger.</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>
<code>_HAL_COMP_COMP1_EXTI_ENABLE_FALLING_E DGE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Enable the COMP1 EXTI line falling edge trigger.</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>

---

`__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE`**Description:**

- Disable the COMP1 EXTI line falling edge trigger.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE`**Description:**

- Enable the COMP1 EXTI line rising & falling edge trigger.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE`**Description:**

- Disable the COMP1 EXTI line rising & falling edge trigger.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_ENABLE_IT`**Description:**

- Enable the COMP1 EXTI line in interrupt mode.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_DISABLE_IT`**Description:**

- Disable the COMP1 EXTI line in interrupt mode.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_GENERATE_SWIT`**Description:**

- Generate a software interrupt on the COMP1 EXTI line.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_ENABLE_EVENT`**Description:**

- Enable the COMP1 EXTI line in event mode.

**Return value:**

- None

`__HAL_COMP_COMP1_EXTI_DISABLE_EVENT`**Description:**

- Disable the COMP1 EXTI line in event mode.

**Return value:**

- None

**Description:**

- Check whether the COMP1 EXTI line flag is set.

**Return value:**

- RESET: or SET

**Description:**

- Clear the COMP1 EXTI flag.

**Return value:**

- None

**Description:**

- Enable the COMP2 EXTI line rising edge trigger.

**Return value:**

- None

**Description:**

- Disable the COMP2 EXTI line rising edge trigger.

**Return value:**

- None

**Description:**

- Enable the COMP2 EXTI line falling edge trigger.

**Return value:**

- None

**Description:**

- Disable the COMP2 EXTI line falling edge trigger.

**Return value:**

- None

**Description:**

- Enable the COMP2 EXTI line rising & falling edge trigger.

**Return value:**

- None

**Description:**

- Disable the COMP2 EXTI

line rising & falling edge trigger.

**Return value:**

- None

`_HAL_COMP_COMP2_EXTI_ENABLE_IT`

**Description:**

- Enable the COMP2 EXTI line in interrupt mode.

**Return value:**

- None

`_HAL_COMP_COMP2_EXTI_DISABLE_IT`

**Description:**

- Disable the COMP2 EXTI line in interrupt mode.

**Return value:**

- None

`_HAL_COMP_COMP2_EXTI_GENERATE_SWIT`

**Description:**

- Generate a software interrupt on the COMP2 EXTI line.

**Return value:**

- None

`_HAL_COMP_COMP2_EXTI_ENABLE_EVENT`

**Description:**

- Enable the COMP2 EXTI line in event mode.

**Return value:**

- None

`_HAL_COMP_COMP2_EXTI_DISABLE_EVENT`

**Description:**

- Disable the COMP2 EXTI line in event mode.

**Return value:**

- None

`_HAL_COMP_COMP2_EXTI_GET_FLAG`

**Description:**

- Check whether the COMP2 EXTI line flag is set.

**Return value:**

- RESET: or SET

`_HAL_COMP_COMP2_EXTI_CLEAR_FLAG`

**Description:**

- Clear the COMP2 EXTI flag.

**Return value:**

- None

***COMP output to EXTI***

COMP_TRIGGERMODE_NONE	Comparator output triggering no External Interrupt Line
COMP_TRIGGERMODE_IT_RISING	Comparator output triggering External Interrupt Line event with interruption, on rising edge
COMP_TRIGGERMODE_IT_FALLING	Comparator output triggering External Interrupt Line event with interruption, on falling edge
COMP_TRIGGERMODE_IT_RISING_FALLING	Comparator output triggering External Interrupt Line event with interruption, on both rising and falling edges
COMP_TRIGGERMODE_EVENT_RISING	Comparator output triggering External Interrupt Line event only (without interruption), on rising edge
COMP_TRIGGERMODE_EVENT_FALLING	Comparator output triggering External Interrupt Line event only (without interruption), on falling edge
COMP_TRIGGERMODE_EVENT_RISING_FALLING	Comparator output triggering External Interrupt Line event only (without interruption), on both rising and falling edges

***COMP private macros to get EXTI line associated with comparators*****COMP\_GET\_EXTI\_LINE    Description:**

- Get the specified EXTI line for a comparator instance.

**Parameters:**

- \_\_INSTANCE\_\_: specifies the COMP instance.

**Return value:**

- value: of

***COMP Handle Management*****\_HAL\_COMP\_RESET\_HANDLE\_STATE    Description:**

- Reset COMP handle state.

**Parameters:**

- \_\_HANDLE\_\_: COMP handle

**Return value:**

- None

**\_HAL\_COMP\_ENABLE****Description:**

- Enable the specified comparator.

**Parameters:**

- `__HANDLE__`: COMP handle

**Return value:**

- None

`__HAL_COMP_DISABLE`**Description:**

- Disable the specified comparator.

**Parameters:**

- `__HANDLE__`: COMP handle

**Return value:**

- None

`__HAL_COMP_LOCK`**Description:**

- Lock the specified comparator configuration.

**Parameters:**

- `__HANDLE__`: COMP handle

**Return value:**

- None

**Notes:**

- Using this macro induce HAL COMP handle state machine being no more in line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL\_COMP\_Lock)".

`__HAL_COMP_IS_LOCKED`**Description:**

- Check whether the specified comparator is locked.

**Parameters:**

- `__HANDLE__`: COMP handle

**Return value:**

- Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

***COMP input minus (inverting input)***

<code>COMP_INPUT_MINUS_1_4VREFINT</code>	Comparator input minus connected to 1/4 VREFINT (only for COMP instance: COMP2)
<code>COMP_INPUT_MINUS_1_2VREFINT</code>	Comparator input minus connected to 1/2 VREFINT (only for COMP instance: COMP2)
<code>COMP_INPUT_MINUS_3_4VREFINT</code>	Comparator input minus connected to 3/4 VREFINT (only for COMP instance: COMP2)
<code>COMP_INPUT_MINUS_VREFINT</code>	Comparator input minus connected to VrefInt
<code>COMP_INPUT_MINUS_DAC1_CH1</code>	Comparator input minus connected to DAC1

	channel 1 (DAC_OUT1)
COMP_INPUT_MINUS_DAC1_CH2	Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)
COMP_INPUT_MINUS_IO1	Comparator input minus connected to IO1 (pin PA0 for COMP1, pin PA2 for COMP2)
COMP_INPUT_MINUS_IO2	Comparator input minus connected to IO2 (pin PB3 for COMP2) (only for COMP instance: COMP2)

***COMP input plus (non-inverting input)***

COMP_INPUT_PLUS_IO1	Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA3 for COMP2)
COMP_INPUT_PLUS_IO2	Comparator input plus connected to IO2 (pin PB4 for COMP2) (only for COMP instance: COMP2)
COMP_INPUT_PLUS_IO3	Comparator input plus connected to IO3 (pin PA5 for COMP2) (only for COMP instance: COMP2)
COMP_INPUT_PLUS_IO4	Comparator input plus connected to IO4 (pin PB6 for COMP2) (only for COMP instance: COMP2)
COMP_INPUT_PLUS_IO5	Comparator input plus connected to IO5 (pin PB7 for COMP2) (only for COMP instance: COMP2)

***COMP private macros to check input parameters***

IS\_COMP\_WINDOWMODE  
 IS\_COMP\_POWERMODE  
 IS\_COMP\_WINDOWMODE\_INSTANCE  
 IS\_COMP\_INPUT\_PLUS  
 IS\_COMP\_INPUT\_MINUS  
 IS\_COMP1\_LPTIMCONNECTION  
 IS\_COMP2\_LPTIMCONNECTION  
 IS\_COMP2\_LPTIMCONNECTION\_RESTRICTED  
 IS\_COMP\_OUTPUTPOL  
 IS\_COMP\_TRIGGERMODE  
 IS\_COMP\_OUTPUT\_LEVEL

***COMP Low power timer connection definition***

COMP_LPTIMCONNECTION_DISABLED	COMPx signal is gated
COMP_LPTIMCONNECTION_IN1_ENABLED	COMPx signal is connected to LPTIM input 1
COMP_LPTIMCONNECTION_IN2_ENABLED	COMPx signal is connected to LPTIM input 2

***COMP Output Level***

COMP\_OUTPUT\_LEVEL\_LOW  
 COMP\_OUTPUT\_LEVEL\_HIGH

***COMP output Polarity***

COMP\_OUTPUTPOL\_NONINVERTED COMP output on GPIO isn't inverted

COMP\_OUTPUTPOL\_INVERTED COMP output on GPIO is inverted

***COMP power mode***

COMP\_POWERMODE\_MEDIUMSPEED COMP power mode to low power  
(indicated as "high speed" in reference  
manual) (only for COMP instance:  
COMP2)

COMP\_POWERMODE\_ULTRALOWPOWER COMP power mode to ultra low power  
(indicated as "low speed" in reference  
manual) (only for COMP instance:  
COMP2)

***COMP Window Mode***

COMP\_WINDOWMODE\_DISABLE Window mode disable:  
Comparators instances pair  
COMP1 and COMP2 are  
independent

COMP\_WINDOWMODE\_COMP1\_INPUT\_PLUS\_COMMON Window mode enable:  
Comparators instances pair  
COMP1 and COMP2 have  
their input plus connected  
together. The common  
input is COMP1 input plus  
(COMP2 input plus is no  
more accessible).

## 9 HAL COMP Extension Driver

### 9.1 COMPEx Firmware driver API description

#### 9.1.1 COMP peripheral Extended features

Comparing to other previous devices, the COMP interface for STM32L0XX devices contains the following additional features

- Possibility to enable or disable the VREFINT which is used as input to the comparator.

#### 9.1.2 Detailed description of functions

##### **HAL\_COMPEx\_EnableVREFINT**

Function Name	<b>void HAL_COMPEx_EnableVREFINT (void )</b>
Function Description	Enable Vrefint and path to comparator, used by comparator instance COMP2 input based on VrefInt or subdivision of VrefInt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• The equivalent of this function is managed automatically when using function "HAL_COMP_Init()".</li><li>• VrefInt requires a startup time (refer to device datasheet, parameter "TVREFINT").</li></ul>

##### **HAL\_COMPEx\_DisableVREFINT**

Function Name	<b>void HAL_COMPEx_DisableVREFINT (void )</b>
Function Description	Disable Vrefint and path to comparator, used by comparator instance COMP2 input based on VrefInt or subdivision of VrefInt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

## 10 HAL CORTEX Generic Driver

### 10.1 CORTEX Firmware driver registers structures

#### 10.1.1 MPU\_Region\_InitTypeDef

##### Data Fields

- *uint32\_t BaseAddress*
- *uint8\_t Enable*
- *uint8\_t Number*
- *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

- ***uint32\_t MPU\_Region\_InitTypeDef::BaseAddress***  
Specifies the base address of the region to protect.
- ***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**
- ***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***  
This parameter is NOT used but is kept to keep API unified through all families
- ***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**

## 10.2 CORTEX Firmware driver API description

### 10.2.1 How to use this driver

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

This section provide functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M0+ exceptions are managed by CMSIS functions.

1. Enable and Configure the priority of the selected IRQ Channels. The priority can be 0..3. Lower priority values gives higher priority. Priority Order: Lowest priority. Lowest hardware priority (IRQn position).
2. Configure the priority of the selected IRQ Channels using HAL\_NVIC\_SetPriority()
3. Enable the selected IRQ Channels using HAL\_NVIC\_EnableIRQ()

#### 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 (0x03).
  - 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 function HAL\_SYSTICK\_CLKSourceConfig(SYSTICK\_CLKSOURCE\_HCLK\_DIV8) just after the HAL\_SYSTICK\_Config() function call. The HAL\_SYSTICK\_CLKSourceConfig() function is defined inside the stm32l0xx\_hal\_cortex.c file.
- You can change the SysTick IRQ priority by calling the HAL\_NVIC\_SetPriority(SysTick\_IRQn,...) function just after the HAL\_SYSTICK\_Config() function call. The HAL\_NVIC\_SetPriority() call the NVIC\_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL\_SYSTICK\_Config() function
  - Reload Value should not exceed 0xFFFFFFF

### 10.2.2 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\_SetPriority()***

- `HAL_NVIC_EnableIRQ()`
- `HAL_NVIC_DisableIRQ()`
- `HAL_NVIC_SystemReset()`
- `HAL_SYSTICK_Config()`
- `HAL MPU_Disable()`
- `HAL MPU_Enable()`

### 10.2.3 Peripheral Control functions

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

This section contains the following APIs:

- `HAL_NVIC_GetPriority()`
- `HAL_NVIC_SetPendingIRQ()`
- `HAL_NVIC_GetPendingIRQ()`
- `HAL_NVIC_ClearPendingIRQ()`
- `HAL_SYSTICK_CLKSourceConfig()`
- `HAL_SYSTICK_IRQHandler()`
- `HAL_SYSTICK_Callback()`
- `HAL_MPU_ConfigRegion()`

### 10.2.4 Detailed description of functions

#### `HAL_NVIC_SetPriority`

Function Name	<code>void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)</code>
Function Description	Sets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn:</b> External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)</li> <li>• <b>PreemptPriority:</b> The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 3. A lower priority value indicates a higher priority</li> <li>• <b>SubPriority:</b> The subpriority level for the IRQ channel. with stm32l0xx devices, this parameter is a dummy value and it is ignored, because no subpriority supported in Cortex M0+ based products.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### `HAL_NVIC_EnableIRQ`

Function Name	<code>void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)</code>
Function Description	Enables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn:</b> External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**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 stm32l0xx.h file)

**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 MPU\_Disable****Function Name****\_STATIC\_INLINE void HAL\_MPU\_Disable (void )****Function Description**

Disable the MPU.

**Return values**

- None:**

**HAL MPU\_Enable****Function Name****\_STATIC\_INLINE 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\_NVIC\_GetPriority**

Function Name **uint32\_t HAL\_NVIC\_GetPriority (IRQn\_Type IRQn)**

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 (stm32l0xxxx.h))

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 stm32l0xx.h file)

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 stm32l0xx.h file)

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 stm32l0xx.h file)

Return values • **None:**

**HAL\_SYSTICK\_CLKSourceConfig**

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

**HAL\_SYSTICK\_IRQHandler**

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

**HAL\_SYSTICK\_Callback**

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

**HAL MPU\_ConfigRegion**

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

## 10.3 CORTEX Firmware driver defines

### 10.3.1 CORTEX

**CORTEX Exported Constants**

IS\_NVIC\_PREEMPTION\_PRIORITY

IS\_NVIC\_DEVICE\_IRQ

**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

**CORTEX 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

**CORTEX SysTick Clock Source**

SYSTICK\_CLKSOURCE\_HCLK\_DIV8  
SYSTICK\_CLKSOURCE\_HCLK  
IS\_SYSTICK\_CLK\_SOURCE

# 11 HAL CRC Generic Driver

## 11.1 CRC Firmware driver registers structures

### 11.1.1 CRC\_InitTypeDef

#### Data Fields

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

#### Field Documentation

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

- 0xA2B3C4D becomes 0xD458B23C CRC\_INPUTDATA\_INVERSION\_WORD word-wise inversion, 0xA2B3C4D becomes 0xB23CD458
- ***uint32\_t CRC\_InitTypeDef::OutputDataInversionMode***  
This parameter is a value of **CRCEx\_Output\_Data\_Inversion** and specifies output data (i.e. CRC) inversion mode. Can be either  
CRC\_OUTPUTDATA\_INVERSION\_DISABLE no CRC inversion, or  
CRC\_OUTPUTDATA\_INVERSION\_ENABLE CRC 0x11223344 is converted into 0x22CC4488

### 11.1.2 CRC\_HandleTypeDef

#### Data Fields

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

#### Field Documentation

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

## 11.2 CRC Firmware driver API description

### 11.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

1. Initialize the CRC according to the specified parameters in the CRC\_InitTypeDef and create the associated handle
2. DeInitialize the CRC peripheral
3. Initialize the CRC MSP
4. DeInitialize CRC MSP

This section contains the following APIs:

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

### 11.2.2 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

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

### 11.2.3 Peripheral State functions

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

This section contains the following APIs:

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

### 11.2.4 Detailed description of functions

#### **HAL\_CRC\_Init**

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

#### **HAL\_CRC\_DelInit**

Function Name	<b>HAL_StatusTypeDef HAL_CRC_DelInit (CRC_HandleTypeDef * hcrc)</b>
Function Description	Deinitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### **HAL\_CRC\_MspInit**

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

**HAL\_CRC\_MspDelInit**

Function Name      **void HAL\_CRC\_MspDelInit (CRC\_HandleTypeDef \* hcrc)**

Function Description      Delinitializes the CRC MSP.

Parameters      •    **hcrc:** CRC handle

Return values      •    **None:**

**HAL\_CRC\_Accumulate**

Function Name      **uint32\_t HAL\_CRC\_Accumulate (CRC\_HandleTypeDef \* hcrc,  
                      uint32\_t pBuffer, uint32\_t BufferLength)**

Function Description      Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.

Parameters      •    **hcrc:** CRC handle  
•    **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.  
•    **BufferLength:** input data buffer length (number of bytes if pBuffer type is \* uint8\_t, number of half-words if pBuffer type is \* uint16\_t, number of words if pBuffer type is \* uint32\_t).

Return values      •    **uint32\_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes      •    By default, the API expects a uint32\_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32\_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

**HAL\_CRC\_Calculate**

Function Name      **uint32\_t HAL\_CRC\_Calculate (CRC\_HandleTypeDef \* hcrc,  
                      uint32\_t pBuffer, uint32\_t BufferLength)**

Function Description      Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

Parameters      •    **hcrc:** CRC handle  
•    **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.  
•    **BufferLength:** input data buffer length (number of bytes if pBuffer type is \* uint8\_t, number of half-words if pBuffer type is \* uint16\_t, number of words if pBuffer type is \* uint32\_t).

Return values      •    **uint32\_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes      •    By default, the API expects a uint32\_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32\_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

**HAL\_CRC\_GetState**

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

## 11.3 CRC Firmware driver defines

### 11.3.1 CRC

*Default CRC computation initialization value*

DEFAULT\_CRC\_INITVALUE

*Indicates whether or not default init value is used*

DEFAULT\_INIT\_VALUE\_ENABLE

DEFAULT\_INIT\_VALUE\_DISABLE

*Indicates whether or not default polynomial is used*

DEFAULT\_POLYNOMIAL\_ENABLE

DEFAULT\_POLYNOMIAL\_DISABLE

*Default CRC generating polynomial*

DEFAULT\_CRC32\_POLY

**CRC Exported Constants**

HAL\_CRC\_Input\_Data\_Reverse

HAL\_CRC\_Output\_Data\_Reverse

**CRC Exported Macros**

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

**Description:**

- Reset CRC handle state.

**Parameters:**

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

**Return value:**

- None

\_\_HAL\_CRC\_DR\_RESET

**Description:**

- Reset CRC Data Register.

**Parameters:**

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

**Return value:**

- None.

\_\_HAL\_CRC\_INITIALCRCVALUE\_CONFIG

**Description:**

- Set CRC INIT non-default value.

**Parameters:**

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

**Return value:**

- None.

`_HAL_CRC_SET_IDR`

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: CRC handle
- \_\_VALUE\_\_: 8-bit value to be stored in the ID register

**Return value:**

- None

`_HAL_CRC_GET_IDR`

**Description:**

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

**Parameters:**

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

**Return value:**

- 8-bit: value of the ID register

***CRC input buffer format***

`CRC_INPUTDATA_FORMAT_UNDEFINED`

`CRC_INPUTDATA_FORMAT_BYTES`

`CRC_INPUTDATA_FORMAT_HALFWORDS`

`CRC_INPUTDATA_FORMAT_WORDS`

***Polynomial sizes to configure the IP***

`CRC_POLYLENGTH_32B`

`CRC_POLYLENGTH_16B`

`CRC_POLYLENGTH_8B`

`CRC_POLYLENGTH_7B`

***CRC polynomial possible sizes actual definitions***

`HAL_CRC_LENGTH_32B`

`HAL_CRC_LENGTH_16B`

`HAL_CRC_LENGTH_8B`

`HAL_CRC_LENGTH_7B`

## 12 HAL CRC Extension Driver

### 12.1 CRCEEx Firmware driver API description

#### 12.1.1 CRC Extended features functions

This subsection provides function allowing to:

- Set CRC polynomial if different from default one.

This section contains the following APIs:

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

#### 12.1.2 Detailed description of functions

##### **HAL\_CRCEEx\_Polynomial\_Set**

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

##### **HAL\_CRCEEx\_Input\_Data\_Reverse**

Function Name	<b>HAL_StatusTypeDef HAL_CRCEEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)</b>
Function Description	Set the Reverse Input data mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> <li>• <b>InputReverseMode:</b> Input Data inversion mode This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– CRC_INPUTDATA_INVERSION_NONE: no change in</li> </ul> </li> </ul>

- bit order (default value)
- CRC\_INPUTDATA\_INVERSION\_BYTE: Byte-wise bit reversal
- CRC\_INPUTDATA\_INVERSION\_HALFWORD: HalfWord-wise bit reversal
- CRC\_INPUTDATA\_INVERSION\_WORD: Word-wise bit reversal

Return values

- **HAL:** status

### **HAL\_CRCEx\_Output\_Data\_Reverse**

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

Function Description      Set the Reverse Output data mode.

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

Return values

- **HAL:** status

## **12.2 CRCEEx Firmware driver defines**

### **12.2.1 CRCEEx**

#### ***CRCEEx Exported Macros***

<b><u>_HAL_CRC_OUTPUTREVERSAL_ENABLE</u></b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Set CRC output reversal.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <u>_HANDLE_</u>: : CRC handle</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None.</li> </ul>
<b><u>_HAL_CRC_OUTPUTREVERSAL_DISABLE</u></b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Unset CRC output reversal.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <u>_HANDLE_</u>: : CRC handle</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None.</li> </ul>
<b><u>_HAL_CRC_POLYNOMIAL_CONFIG</u></b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Set CRC non-default polynomial.</li> </ul>
	<b>Parameters:</b>

- `__HANDLE__`: : CRC handle
- `__POLYNOMIAL__`: 7, 8, 16 or 32-bit polynomial

**Return value:**

- None.

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

`CRC_INPUTDATA_INVERSION_NONE`  
`CRC_INPUTDATA_INVERSION_BYTE`  
`CRC_INPUTDATA_INVERSION_HALFWORD`  
`CRC_INPUTDATA_INVERSION_WORD`

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

`CRC_OUTPUTDATA_INVERSION_DISABLE`  
`CRC_OUTPUTDATA_INVERSION_ENABLE`

## 13 HAL CRYP Generic Driver

### 13.1 CRYP Firmware driver registers structures

#### 13.1.1 CRYP\_InitTypeDef

##### Data Fields

- *uint32\_t DataType*
- *uint8\_t \* pKey*
- *uint8\_t \* pInitVect*

##### Field Documentation

- ***uint32\_t CRYP\_InitTypeDef::DataType***  
32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP\_Data\_Type***
- ***uint8\_t\* CRYP\_InitTypeDef::pKey***  
The key used for encryption/decryption
- ***uint8\_t\* CRYP\_InitTypeDef::pInitVect***  
The initialization vector used also as initialization counter in CTR mode

#### 13.1.2 CRYP\_HandleTypeDefDef

##### Data Fields

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

##### Field Documentation

- ***AES\_TypeDef\* CRYP\_HandleTypeDefDef::Instance***  
Register base address
- ***CRYP\_InitTypeDef CRYP\_HandleTypeDefDef::Init***  
CRYP required parameters
- ***uint8\_t\* CRYP\_HandleTypeDefDef::pCrypInBuffPtr***  
Pointer to CRYP processing (encryption, decryption,...) buffer

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

## 13.2 CRYP Firmware driver API description

### 13.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYP according to the specified parameters in the CRYP\_InitTypeDef and creates the associated handle
- DeInitialize the CRYP peripheral
- Initialize the CRYP MSP
- DeInitialize CRYP MSP

This section contains the following APIs:

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

### 13.2.2 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES algorithm in different chaining modes
- Decrypt ciphertext using AES algorithm in different chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

- [\*\*\*HAL\\_CRYP\\_AESECB\\_Encrypt\(\)\*\*\*](#)
- [\*\*\*HAL\\_CRYP\\_AESCBC\\_Encrypt\(\)\*\*\*](#)
- [\*\*\*HAL\\_CRYP\\_AESCTR\\_Encrypt\(\)\*\*\*](#)
- [\*\*\*HAL\\_CRYP\\_AESECB\\_Decrypt\(\)\*\*\*](#)

- [`HAL\_CRYP\_AESCBC\_Decrypt\(\)`](#)
- [`HAL\_CRYP\_AESCTR\_Decrypt\(\)`](#)
- [`HAL\_CRYP\_AESECB\_Encrypt\_IT\(\)`](#)
- [`HAL\_CRYP\_AESCBC\_Encrypt\_IT\(\)`](#)
- [`HAL\_CRYP\_AESCTR\_Encrypt\_IT\(\)`](#)
- [`HAL\_CRYP\_AESECB\_Decrypt\_IT\(\)`](#)
- [`HAL\_CRYP\_AESCBC\_Decrypt\_IT\(\)`](#)
- [`HAL\_CRYP\_AESCTR\_Decrypt\_IT\(\)`](#)
- [`HAL\_CRYP\_AESECB\_Encrypt\_DMA\(\)`](#)
- [`HAL\_CRYP\_AESCBC\_Encrypt\_DMA\(\)`](#)
- [`HAL\_CRYP\_AESCTR\_Encrypt\_DMA\(\)`](#)
- [`HAL\_CRYP\_AESECB\_Decrypt\_DMA\(\)`](#)
- [`HAL\_CRYP\_AESCBC\_Decrypt\_DMA\(\)`](#)
- [`HAL\_CRYP\_AESCTR\_Decrypt\_DMA\(\)`](#)

### 13.2.3 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

This section contains the following APIs:

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

### 13.2.4 Peripheral State functions

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

This section contains the following APIs:

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

### 13.2.5 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error

This section contains the following APIs:

- [`HAL\_CRYP\_ErrorCallback\(\)`](#)
- [`HAL\_CRYP\_InCpltCallback\(\)`](#)
- [`HAL\_CRYP\_OutCpltCallback\(\)`](#)

### 13.2.6 Detailed description of functions

#### `HAL_CRYP_Init`

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

**HAL\_CRYP\_DelInit**

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

**HAL\_CRYP\_MspInit**

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

**HAL\_CRYP\_MspDelInit**

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

**HAL\_CRYP\_AESECB\_Encrypt**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Timeout:</b> Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_CRYP\_AESECB\_Decrypt**

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

	Parameters	Parameters
		• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32) • <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16. • <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32) • <b>Timeout:</b> Specify Timeout value
	Return values	• <b>HAL:</b> status

### HAL\_CRYP\_AESCBC\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Timeout:</b> Specify Timeout value</li> </ul>
Return values	• <b>HAL:</b> status

### HAL\_CRYP\_AESCBC\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Timeout:</b> Specify Timeout value</li> </ul>
Return values	• <b>HAL:</b> status

### HAL\_CRYP\_AESCTR\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode then encrypt pPlainData.

Parameters	<ul style="list-style-type: none"> <li><b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li><b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li><b>Timeout:</b> Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_CRYP\_AESCTR\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li><b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li><b>Timeout:</b> Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

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

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li><b>Size:</b> Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li><b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

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

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

- contains the configuration information for CRYP module
  - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
  - **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
  - **pCypherData:** Pointer to the ciphertext buffer (aligned on u32)
- Return values
- **HAL:** status

### **HAL\_CRYP\_AESCTR\_Encrypt\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_CRYP\_AESECB\_Decrypt\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_CRYP\_AESCTR\_Decrypt\_IT**

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

---

	u32)
	<ul style="list-style-type: none"> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

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

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES CBC decryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

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

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

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

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16 bytes</li> </ul>

- Return values
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
  - **HAL:** status

### **HAL\_CRYP\_AESCBC\_Encrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_CRYP\_AESCBC\_Decrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES CBC decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_CRYP\_AESCTR\_Encrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_CRYP\_AESCTR\_Decrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData:</b> Pointer to the ciphertext buffer (aligned on u32)</li> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer (aligned on u32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_CRYP\_InCpltCallback**

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

**HAL\_CRYP\_OutCpltCallback**

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

**HAL\_CRYP\_ErrorCallback**

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

**HAL\_CRYP\_IRQHandler**

Function Name	<b>void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)</b>
Function Description	This function handles CRYP interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYP_HandleTypeDef structure that</li> </ul>

contains the configuration information for CRYP module

Return values

- **None:**

### **HAL\_CRYP\_GetState**

Function Name

**HAL\_CRYP\_STATETypeDef HAL\_CRYP\_GetState  
(CRYP\_HandleTypeDef \* hcryp)**

Function Description

Returns the CRYP state.

Parameters

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

Return values

- **HAL:** state

## **13.3 CRYP Firmware driver defines**

### **13.3.1 CRYP**

#### ***AES Clear Flags***

**CRYP\_CLEARFLAG\_CCF** Computation Complete Flag Clear

**CRYP\_CLEARFLAG\_RDERR** Read Error Clear

**CRYP\_CLEARFLAG\_WRERR** Write Error Clear

#### ***AES Flags***

**CRYP\_FLAG\_CCF** Computation Complete Flag

**CRYP\_FLAG\_RDERR** Read Error Flag

**CRYP\_FLAG\_WRERR** Write Error Flag

#### ***AES Interrupts***

**CRYP\_IT\_CC** Computation Complete interrupt

**CRYP\_IT\_ERR** Error interrupt

#### ***CRYP Algo Mode Direction***

**CRYP\_CR\_ALGOMODE\_DIRECTION**

**CRYP\_CR\_ALGOMODE\_AES\_ECB\_ENCRYPT**

**CRYP\_CR\_ALGOMODE\_AES\_ECB\_KEYDERDECRYPT**

**CRYP\_CR\_ALGOMODE\_AES\_CBC\_ENCRYPT**

**CRYP\_CR\_ALGOMODE\_AES\_CBC\_KEYDERDECRYPT**

**CRYP\_CR\_ALGOMODE\_AES\_CTR\_ENCRYPT**

**CRYP\_CR\_ALGOMODE\_AES\_CTR\_DECRYPT**

#### ***CRYP Data Type***

**CRYP\_DATATYPE\_32B**

**CRYP\_DATATYPE\_16B**

**CRYP\_DATATYPE\_8B**

**CRYP\_DATATYPE\_1B**

`IS_CRYP_DATATYPE`***CRYP Exported Macros***`_HAL_CRYP_RESET_HANDLE_STATE`**Description:**

- Reset CRYP handle state.

**Parameters:**

- `_HANDLE_`: specifies the CRYP handle.

**Return value:**

- None

`_HAL_CRYP_ENABLE`**Description:**

- Enable/Disable the CRYP peripheral.

**Parameters:**

- `_HANDLE_`: specifies the CRYP handle.

**Return value:**

- None

`_HAL_CRYP_DISABLE``_HAL_CRYP_SET_MODE`**Description:**

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

**Parameters:**

- `_HANDLE_`: specifies the CRYP handle.
- `_MODE_`: The algorithm mode.

**Return value:**

- None

`_HAL_CRYP_GET_FLAG`**Description:**

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

**Parameters:**

- `_HANDLE_`: specifies the CRYP handle.
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - CRYP\_FLAG\_CCF : Computation Complete Flag
  - CRYP\_FLAG\_RDERR : Read Error Flag
  - CRYP\_FLAG\_WRERR : Write Error Flag

[\\_\\_HAL\\_CRYP\\_CLEAR\\_FLAG](#)**Return value:**

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

**Description:**

- Clear the CRYP pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CRYP handle.
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one of the following values:
  - CRYP\_CLEARFLAG\_CCF : Computation Complete Clear Flag
  - CRYP\_CLEARFLAG\_RDERR : Read Error Clear
  - CRYP\_CLEARFLAG\_WRERR : Write Error Clear

**Return value:**

- None

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

- Enable the CRYP interrupt.

**Parameters:**

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

**Return value:**

- None

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

- Disable the CRYP interrupt.

**Parameters:**

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

**Return value:**

- None

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

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CRYP handle.
- \_\_INTERRUPT\_\_: CRYP interrupt source

to check This parameter can be one of the following values:

- CRYP\_IT\_CC : Computation Complete interrupt
- CRYP\_IT\_ERR : Error interrupt (used for RDERR and WRERR)

**Return value:**

- State: of interruption (SET or RESET)

[\\_\\_HAL\\_CRYP\\_CLEAR\\_IT](#)

**Description:**

- Clear the CRYP pending IT.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the CRYP handle.
- [\\_\\_IT\\_\\_](#): specifies the IT to clear. This parameter can be one of the following values:
  - CRYP\_CLEARFLAG\_CCF : Computation Complete Clear Flag
  - CRYP\_CLEARFLAG\_RDERR : Read Error Clear
  - CRYP\_CLEARFLAG\_WRERR : Write Error Clear

**Return value:**

- None

## 14 HAL CRYP Extension Driver

### 14.1 CRYPEx Firmware driver API description

#### 14.1.1 Extended features functions

This section provides callback functions:

- Computation completed.

This section contains the following APIs:

- [\*HAL\\_CRYPEx\\_ComputationCpltCallback\(\)\*](#)

#### 14.1.2 Detailed description of functions

##### **HAL\_CRYPEx\_ComputationCpltCallback**

Function Name      **void HAL\_CRYPEx\_ComputationCpltCallback  
(CRYP\_HandleTypeDef \* hcryp)**

Function Description      Computation completed callbacks.

Parameters      • **hcryp:** pointer to a CRYP\_HandleTypeDef structure that contains the configuration information for CRYP module

Return values      • **None:**

## 15 HAL DAC Generic Driver

### 15.1 DAC Firmware driver registers structures

#### 15.1.1 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

#### 15.1.2 DAC\_ChannelConfTypeDef

##### Data Fields

- *uint32\_t DAC\_Trigger*
- *uint32\_t DAC\_OutputBuffer*

##### Field Documentation

- ***uint32\_t DAC\_ChannelConfTypeDef::DAC\_Trigger***  
Specifies the external trigger for the selected DAC channel. This parameter can be a value of [\*\*DAC\\_trigger\\_selection\*\*](#)
- ***uint32\_t DAC\_ChannelConfTypeDef::DAC\_OutputBuffer***  
Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of [\*\*DAC\\_output\\_buffer\*\*](#)

## 15.2 DAC Firmware driver API description

### 15.2.1 DAC Peripheral features

#### DAC Channels

STM32F0 devices integrates no, one or two 12-bit Digital Analog Converters. STM32L05x & STM32L06x devices have one converter (channel1) STM32L07x & STM32L08x devices have two converters (i.e. channel1 & channel2) When 2 converters are present (i.e. channel1 & channel2) they 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 (STM32L07x/STM32L08x only)
3. Channel1 & channel2 can be used independently or simultaneously in dual mode (STM32L07x/STM32L08x only)

#### DAC Triggers

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

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx\_Pin9) using DAC\_Trigger\_Ext\_IT9. The used pin (GPIOx\_Pin9) must be configured in input mode.
2. Timers TRGO: STM32L05x/STM32L06x : TIM2, TIM6 and TIM21  
STM32L07x/STM32L08x : TIM2, TIM3, TIM6, TIM7 and TIM21  
(DAC\_Trigger\_T2\_TRGO, DAC\_Trigger\_T6\_TRGO...)
3. Software using DAC\_Trigger\_Software

#### DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC\_OutputBuffer = DAC\_OutputBuffer\_Enable;



Refer to the device datasheet for more details about output impedance value with and without output buffer.

#### DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave using HAL\_DACEx\_NoiseWaveGenerate()
2. Triangle wave using HAL\_DACEx\_TriangleWaveGenerate()

#### DAC data format

The DAC data format can be:

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

### DAC data value to voltage correspondence

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

### DMA requests

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

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 Request9 channel2 which must be already configured
2. DAC channel2 : mapped on DMA1 Request15 channel4 which must be already configured (STM32L07x/STM32L08x only) For Dual mode (STM32L07x/STM32L08x only) and specific signal Triangle and noise generation please refer to Extension Features Driver description

## 15.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL\_DAC\_Init()
- Configure DAC\_OUT1: PA4 in analog mode.
- Configure DAC\_OUT2: PA5 in analog mode (STM32L07x/STM32L08x only).
- Configure the DAC channel using HAL\_DAC\_ConfigChannel() function.
- Enable the DAC channel using HAL\_DAC\_Start() or HAL\_DAC\_Start\_DMA functions

### Polling mode IO operation

- Start the DAC peripheral using HAL\_DAC\_Start()
- To read the DAC last data output value value, use the HAL\_DAC\_GetValue() function.
- Stop the DAC peripheral using HAL\_DAC\_Stop()

### DMA mode IO operation

- Start the DAC peripheral using HAL\_DAC\_Start\_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At the middle of data transfer HAL\_DAC\_ConvHalfCpltCallbackCh1() or HAL\_DAC\_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() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ErrorCallbackCh1

- In case of DMA underrun, DAC interruption triggers and execute internal function HAL\_DAC\_IRQHandler. HAL\_DAC\_DMAUnderrunCallbackCh1() or HAL\_DAC\_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_DMAUnderrunCallbackCh1 or HAL\_DAC\_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
- `_HAL_DAC_DISABLE` : Disable the DAC peripheral
- `_HAL_DAC_CLEAR_FLAG`: Clear the DAC's pending flags
- `_HAL_DAC_GET_FLAG`: Get the selected DAC's flag status



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

### 15.2.3 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()`

### 15.2.4 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.
- Get result of dual mode conversion (STM32L07xx/STM32L08xx only)

This section contains the following APIs:

- `HAL_DAC_Start()`
- `HAL_DAC_Stop()`
- `HAL_DAC_Start_DMA()`
- `HAL_DAC_Stop_DMA()`
- `HAL_DAC_GetValue()`
- `HAL_DAC_IRQHandler()`
- `HAL_DAC_ConvCpltCallbackCh1()`
- `HAL_DAC_ConvHalfCpltCallbackCh1()`
- `HAL_DAC_ErrorCallbackCh1()`

- [\*HAL\\_DAC\\_DMAUnderrunCallbackCh1\(\)\*](#)
- [\*HAL\\_DAC\\_SetValue\(\)\*](#)

### 15.2.5 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\(\)\*](#)

### 15.2.6 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\\_SetValue\(\)\*](#)

### 15.2.7 Detailed description of functions

#### **HAL\_DAC\_Init**

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

#### **HAL\_DAC\_DeInit**

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

#### **HAL\_DAC\_MspInit**

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

contains the configuration information for the specified DAC.

Return values

- **None:**

### **HAL\_DAC\_MspDelInit**

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

Function Description      Delinitializes 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  
(STM32L07x/STM32L08x only)

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 using DMA.

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  
(STM32L07x/STM32L08x only)
  - **pData:** The destination peripheral Buffer address.
  - **Length:** The length of data to be transferred from memory to DAC peripheral
  - **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
    - DAC\_ALIGN\_8B\_R: 8bit right data alignment 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        | <b>HAL_StatusTypeDef HAL_DAC_Stop_DMA<br/>(DAC_HandleTypeDef * hdac, uint32_t Channel)</b>  |
| Function Description | Disables DAC and stop conversion of channel.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel:</b> The selected DAC channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- DAC_CHANNEL_1: DAC Channel1 selected</li> <li>- DAC_CHANNEL_2: DAC Channel2 selected<br/>(STM32L07x/STM32L08x only)</li> </ul> </li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>  |

### HAL\_DAC\_SetValue

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_DAC_SetValue<br/>(DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)</b>   |
| Function Description | Set the specified data holding register value for DAC channel.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel:</b> The selected DAC channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- DAC_CHANNEL_1: DAC Channel1 selected</li> <li>- DAC_CHANNEL_2: DAC Channel2 selected<br/>(STM32L07x/STM32L08x only)</li> </ul> </li> <li>• <b>Alignment:</b> Specifies the data alignment. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- DAC_ALIGN_8B_R: 8bit right data alignment selected</li> <li>- DAC_ALIGN_12B_L: 12bit left data alignment selected</li> <li>- DAC_ALIGN_12B_R: 12bit right data alignment selected</li> </ul> </li> <li>• <b>Data:</b> Data to be loaded in the selected data holding register.</li> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel:</b> The selected DAC channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- DAC_CHANNEL_1: DAC Channel1 selected</li> </ul> </li> </ul> |

- 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:** status

### **HAL\_DAC\_GetValue**

Function Name	<b>uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)</b>
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel:</b> The selected DAC channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– DAC_CHANNEL_1: DAC Channel1 selected</li> <li>– DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>The:</b> selected DAC channel data output value.</li> </ul>

### **HAL\_DAC\_IRQHandler**

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

### **HAL\_DAC\_ConvCpltCallbackCh1**

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

### **HAL\_DAC\_ConvHalfCpltCallbackCh1**

Function Name	<b>void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)</b>
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel1.

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

### HAL\_DAC\_ErrorCallbackCh1

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

### HAL\_DAC\_DMADebugCallbackCh1

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

### HAL\_DAC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)</b>
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>sConfig:</b> DAC configuration structure.</li> <li>• <b>Channel:</b> The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– DAC_CHANNEL_1: DAC Channel1 selected</li> <li>– DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_DAC\_GetState

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

**HAL\_DAC\_GetError**

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

## 15.3 DAC Firmware driver defines

### 15.3.1 DAC

***DAC Channel selection***`DAC_CHANNEL_1``DAC_CHANNEL_2``IS_DAC_CHANNEL`***DAC data***`IS_DAC_DATA`***DAC data alignment***`DAC_ALIGN_12B_R``DAC_ALIGN_12B_L``DAC_ALIGN_8B_R``IS_DAC_ALIGN`***DAC Error Code***`HAL_DAC_ERROR_NONE` No error`HAL_DAC_ERROR_DMAUNDERRUNCH1` DAC channel1 DAM underrun error`HAL_DAC_ERROR_DMAUNDERRUNCH2` DAC channel2 DAM underrun error`HAL_DAC_ERROR_DMA` DMA error***DAC Exported Macros***`_HAL_DAC_RESET_HANDLE_STATE` **Description:**

- Reset DAC handle state.

**Parameters:**

- `_HANDLE_`: specifies the DAC handle.

**Return value:**

- None

`_HAL_DAC_ENABLE` **Description:**

- Enable the DAC channel.

**Parameters:**

- `_HANDLE_`: specifies the DAC handle.
- `_DAC_CHANNEL_`: specifies the DAC

channel

**Return value:**

- None

`__HAL_DAC_DISABLE`

**Description:**

- Disable the DAC channel.

**Parameters:**

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

**Return value:**

- None

`__HAL_DAC_ENABLE_IT`

**Description:**

- Enable the DAC interrupt.

**Parameters:**

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

**Return value:**

- None

`__HAL_DAC_GET_IT_SOURCE`

**Description:**

- Check whether the specified DAC interrupt source is enabled or not.

**Parameters:**

- `__HANDLE__`: DAC handle
- `__INTERRUPT__`: DAC interrupt source to check This parameter can be any combination of the following values:
  - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
  - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt  
(STM32L072xx STM32L073xx  
STM32L082xx STM32L083xx only)

**Return value:**

- State: of interruption (SET or RESET)

`__HAL_DAC_GET_FLAG`

**Description:**

- Get the selected DAC's flag status.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the FLAG.

**Return value:**

- None

`__HAL_DAC_CLEAR_FLAG`**Description:**

- Clear the DAC's flag.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the FLAG.

**Return value:**

- None

***DAC flags definition***`DAC_FLAG_DMAUDR1``DAC_FLAG_DMAUDR2`***DAC IT definition***`DAC_IT_DMAUDR1``DAC_IT_DMAUDR2`***DAC output buffer***`DAC_OUTPUTBUFFER_ENABLE``DAC_OUTPUTBUFFER_DISABLE``IS_DAC_OUTPUT_BUFFER_STATE`***DAC trigger selection***

<code>DAC_TRIGGER_NONE</code>	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
-------------------------------	---

<code>DAC_TRIGGER_T6_TRGO</code>	TIM6 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_T21_TRGO</code>	TIM21 TRGO selected as external conversion trigger for DAC channel
-----------------------------------	--

<code>DAC_TRIGGER_T2_TRGO</code>	TIM2 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_EXT_IT9</code>	EXTI Line9 event selected as external conversion trigger for DAC channel
----------------------------------	--

<code>DAC_TRIGGER_SOFTWARE</code>	Conversion started by software trigger for DAC channel
-----------------------------------	--

<code>DAC_TRIGGER_T3_TRGO</code>	TIM3 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_T3_CH3</code>	TIM3 CH3 selected as external conversion trigger for DAC channel
---------------------------------	--

<code>DAC_TRIGGER_T7_TRGO</code>	TIM7 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

`IS_DAC_TRIGGER`

## 16 HAL DAC Extension Driver

### 16.1 DACEx Firmware driver API description

#### 16.1.1 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.

#### 16.1.2 Detailed description of functions

##### **HAL\_DACEx\_TriangleWaveGenerate**

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)</b>
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel:</b> The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- DAC_CHANNEL_1: DAC Channel1 selected</li> <li>- DAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)</li> </ul> </li> <li>• <b>Amplitude:</b> Select max triangle amplitude. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1</li> <li>- DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3</li> <li>- DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7</li> <li>- DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15</li> <li>- DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31</li> <li>- DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63</li> <li>- DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127</li> <li>- DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255</li> <li>- DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511</li> <li>- DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023</li> </ul> </li> </ul>

- 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 Channel1 selected
  - DAC\_CHANNEL\_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
- **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\_DualGetValue**

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

**HAL\_DACEx\_DualSetValue**

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)</code>
Function Description	Set the specified data holding register value for dual DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Alignment:</b> Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected</li> <li>• <b>Data1:</b> Data for DAC Channel2 to be loaded in the selected data holding register.</li> <li>• <b>Data2:</b> Data for DAC Channel1 to be loaded in the selected data holding register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In dual mode, a unique register access is required to write in both DAC channels at the same time.</li> </ul>

**HAL\_DACEx\_ConvCpltCallbackCh2**

Function Name	<code>void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)</code>
Function Description	Conversion complete callback in non blocking mode for Channel2.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_DACEx\_ConvHalfCpltCallbackCh2**

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

## Return values

- **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:**

**HAL\_DACEx\_DMAUnderrunCallbackCh2**

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

Function Description      DMA underrun DAC callback for channel2.

Parameters     
 

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

Return values     
 

- **None:**

## 16.2 DACEx Firmware driver defines

### 16.2.1 DACEx

**DACEx Ifsrnmask 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
IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE	

## 17 HAL DMA Generic Driver

### 17.1 DMA Firmware driver registers structures

#### 17.1.1 DMA\_InitTypeDef

##### Data Fields

- *uint32\_t Request*
- *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::Request***  
Specifies the request selected for the specified channel. This parameter can be a value of [DMA\\_request](#)
- ***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. When Memory to Memory transfer is used, this is the Source Increment mode 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. When Memory to Memory transfer is used, this is the Destination Increment mode This parameter can be a value of [DMA\\_Memory\\_incremented\\_mode](#)
- ***uint32\_t DMA\_InitTypeDef::PeriphDataAlignment***  
Specifies the Peripheral data width. When Memory to Memory transfer is used, this is the Source Alignment format This parameter can be a value of [DMA\\_Peripheral\\_data\\_size](#)
- ***uint32\_t DMA\_InitTypeDef::MemDataAlignment***  
Specifies the Memory data width. When Memory to Memory transfer is used, this is the Destination Alignment format This parameter can be a value of [DMA\\_Memory\\_data\\_size](#)
- ***uint32\_t DMA\_InitTypeDef::Mode***  
Specifies the operation mode of the DMAy Channelx (Normal or Circular). 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](#)

### 17.1.2 `__DMA_HandleTypeDef`

#### Data Fields

- `DMA_Channel_TypeDef * Instance`
- `DMA_InitTypeDef Init`
- `HAL_LockTypeDef Lock`
- `__IO HAL_DMA_StateTypeDef State`
- `void * Parent`
- `void(* XferCpltCallback`
- `void(* XferHalfCpltCallback`
- `void(* XferErrorCallback`
- `__IO uint32_t ErrorCode`

#### 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
- **`__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State`**  
DMA transfer state
- **`void* __DMA_HandleTypeDef::Parent`**  
Parent object state
- **`void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)`**  
DMA transfer complete callback
- **`void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)`**  
DMA Half transfer complete callback
- **`void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)`**  
DMA transfer error callback
- **`__IO uint32_t __DMA_HandleTypeDef::ErrorCode`**  
DMA Error code

## 17.2 DMA Firmware driver API description

### 17.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DMA
- De-Initialize the DMA

This section contains the following APIs:

- [`HAL\_DMA\_Init\(\)`](#)
- [`HAL\_DMA\_DeInit\(\)`](#)

## 17.2.2 IO operation functions

This section provides functions allowing to:

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

This section contains the following APIs:

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

## 17.2.3 Peripheral State 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\(\)\*](#)

## 17.2.4 Detailed description of functions

### **HAL\_DMA\_Init**

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

### **HAL\_DMA\_DeInit**

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

**HAL\_DMA\_Start**

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

**HAL\_DMA\_Start\_IT**

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

**HAL\_DMA\_Abort**

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

**HAL\_DMA\_PollForTransfer**

Function Name	<b>HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)</b>
Function Description	Polling for transfer complete.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</b> pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.</li> </ul>

- **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\_GetState**

- Function Name      **HAL\_DMA\_StateTypeDef HAL\_DMA\_GetState (DMA\_HandleTypeDef \* hdma)**
- Function Description      Returns the DMA 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

## **17.3 DMA Firmware driver defines**

### **17.3.1 DMA**

#### **DMA Data Buffer Size Check**

**IS\_DMA\_BUFFER\_SIZE**

#### **DMA Data Transfer directions**

**DMA\_PERIPH\_TO\_MEMORY**      Peripheral to memory direction

**DMA\_MEMORY\_TO\_PERIPH**      Memory to peripheral direction

**DMA\_MEMORY\_TO\_MEMORY**      Memory to memory direction

**IS\_DMA\_DIRECTION**

#### **DMA Error Codes**

---

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_TIMEOUT	Timeout error
IS_DMA_ALL_INSTANCE	
IS_DMA_ALL_CONTROLLER	

**DMA Exported Macros**

<code>_HAL_DMA_RESET_HANDLE_STATE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Reset DMA handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: DMA handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• None</li> </ul>
<code>_HAL_DMA_ENABLE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Enable the specified DMA Channel.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: DMA handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• None.</li> </ul>
<code>_HAL_DMA_DISABLE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Disable the specified DMA Channel.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: DMA handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• None.</li> </ul>
<code>_HAL_DMA_GET_TC_FLAG_INDEX</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Returns the current DMA Channel transfer complete flag.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: DMA handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• The specified transfer complete flag index.</li> </ul>
<code>_HAL_DMA_GET_HT_FLAG_INDEX</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Returns the current DMA Channel half transfer complete flag.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: DMA handle</li> </ul>

**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:**

- Returns 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\_TCIFx: Transfer complete flag
  - DMA\_FLAG\_HTIFx: Half transfer complete flag
  - DMA\_FLAG\_TEIFx: Transfer error flag
  - DMA\_ISR\_GIFx: Global interrupt flag Where x can be 0\_4, 1\_5, 2\_6 or 3\_7 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\_TCIFx: Transfer complete flag
- DMA\_FLAG\_HTIFx: Half transfer complete flag
- DMA\_FLAG\_TEIFx: Transfer error flag
- DMA\_ISR\_GIFx: Global interrupt flag Where x can be 0\_4, 1\_5, 2\_6 or 3\_7 to select the DMA Channel flag.

**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:**

- Disables 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:**

- Checks 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:**

- Returns the number of remaining data units in the current DMAy Channelx 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 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_TRIGGER	Index of the DMA handle used for Trigger DMA requests

***DMA Interrupt Definitions***

DMA\_IT\_TC  
DMA\_IT\_HT  
DMA\_IT\_TE

***DMA Memory Data Size Alignment***

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

IS\_DMA\_MEMORY\_DATA\_SIZE

***DMA Memory Incremented Mode***

DMA_MINC_ENABLE	Memory increment mode Enable
DMA_MINC_DISABLE	Memory increment mode Disable

IS\_DMA\_MEMORY\_INC\_STATE

***DMA Mode***

---

DMA_NORMAL	Normal Mode
DMA_CIRCULAR	Circular Mode
IS_DMA_MODE	

***DMA Peripheral Data Size Alignment***

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

IS\_DMA\_PERIPHERAL\_DATA\_SIZE

***DMA Peripheral Incremented Mode***

DMA_PINC_ENABLE	Peripheral increment mode Enable
DMA_PINC_DISABLE	Peripheral increment mode Disable
IS_DMA_PERIPHERAL_INC_STATE	

***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
IS_DMA_PRIORITY	

***DMA request definitions***

DMA_REQUEST_0	
DMA_REQUEST_1	
DMA_REQUEST_2	
DMA_REQUEST_3	
DMA_REQUEST_4	
DMA_REQUEST_5	
DMA_REQUEST_6	
DMA_REQUEST_7	
DMA_REQUEST_8	
DMA_REQUEST_9	
DMA_REQUEST_10	
DMA_REQUEST_11	
DMA_REQUEST_12	
DMA_REQUEST_13	
DMA_REQUEST_14	
DMA_REQUEST_15	
IS_DMA_ALL_REQUEST	

## 18 HAL FIREWALL Generic Driver

### 18.1 FIREWALL Firmware driver registers structures

#### 18.1.1 FIREWALL\_InitTypeDef

##### Data Fields

- *uint32\_t CodeSegmentStartAddress*
- *uint32\_t CodeSegmentLength*
- *uint32\_t NonVDataSegmentStartAddress*
- *uint32\_t NonVDataSegmentLength*
- *uint32\_t VDataSegmentStartAddress*
- *uint32\_t VDataSegmentLength*
- *uint32\_t VolatileDataExecution*
- *uint32\_t VolatileDataShared*

##### Field Documentation

- ***uint32\_t FIREWALL\_InitTypeDef::CodeSegmentStartAddress***  
Protected code segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- ***uint32\_t FIREWALL\_InitTypeDef::CodeSegmentLength***  
Protected code segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- ***uint32\_t FIREWALL\_InitTypeDef::NonVDataSegmentStartAddress***  
Protected non-volatile data segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- ***uint32\_t FIREWALL\_InitTypeDef::NonVDataSegmentLength***  
Protected non-volatile data segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- ***uint32\_t FIREWALL\_InitTypeDef::VDataSegmentStartAddress***  
Protected volatile data segment start address. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 in order to allow a 64-byte granularity.
- ***uint32\_t FIREWALL\_InitTypeDef::VDataSegmentLength***  
Protected volatile data segment length in bytes. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 for the length to be a multiple of 64 bytes.
- ***uint32\_t FIREWALL\_InitTypeDef::VolatileDataExecution***  
Set VDE bit specifying whether or not the volatile data segment can be executed.  
When VDS = 1 (set by parameter VolatileDataShared), VDE bit has no meaning. This parameter can be a value of ***FIREWALL\_VolatileData\_Executable***
- ***uint32\_t FIREWALL\_InitTypeDef::VolatileDataShared***  
Set VDS bit in specifying whether or not the volatile data segment can be shared with a non-protected application code. This parameter can be a value of ***FIREWALL\_VolatileData\_Shared***

## 18.2 FIREWALL Firmware driver API description

### 18.2.1 How to use this driver

The FIREWALL HAL driver can be used as follows:

1. Declare a FIREWALL\_InitTypeDef initialization structure.
2. Resort to HAL\_FIREWALL\_Config() API to initialize the Firewall
3. Enable the FIREWALL in calling HAL\_FIREWALL\_EnableFirewall() API
4. To ensure that any code executed outside the protected segment closes the FIREWALL, the user must set the flag FIREWALL\_PRE\_ARM\_SET in calling \_\_HAL\_FIREWALL\_PREAMP\_ENABLE() macro if called within a protected code segment or HAL\_FIREWALL\_EnablePreArmFlag() API if called outside of protected code segment after HAL\_FIREWALL\_Config() call.

### 18.2.2 Initialization and Configuration functions

This subsection provides the functions allowing to initialize the Firewall. Initialization is done by HAL\_FIREWALL\_Config():

- Enable the Firewall clock thru \_\_HAL\_RCC\_FIREWALL\_CLK\_ENABLE() macro.
- Set the protected code segment address start and length.
- Set the protected non-volatile and/or volatile data segments address starts and lengths if applicable.
- Set the volatile data segment execution and sharing status.
- Length must be set to 0 for an unprotected segment.

This section contains the following APIs:

- [\*\*\*HAL\\_FIREWALL\\_Config\(\)\*\*\*](#)
- [\*\*\*HAL\\_FIREWALL\\_GetConfig\(\)\*\*\*](#)
- [\*\*\*HAL\\_FIREWALL\\_EnableFirewall\(\)\*\*\*](#)
- [\*\*\*HAL\\_FIREWALL\\_EnablePreArmFlag\(\)\*\*\*](#)
- [\*\*\*HAL\\_FIREWALL\\_DisablePreArmFlag\(\)\*\*\*](#)

### 18.2.3 Detailed description of functions

#### ***HAL\_FIREWALL\_Config***

Function Name	<b><i>HAL_StatusTypeDef HAL_FIREWALL_Config (FIREWALL_InitTypeDef * fw_init)</i></b>
Function Description	Initialize the Firewall according to the FIREWALL_InitTypeDef structure parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b><i>fw_init:</i></b> Firewall initialization structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b><i>HAL:</i></b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The API returns HAL_ERROR if the Firewall is already enabled.</li> </ul>

#### ***HAL\_FIREWALL\_GetConfig***

Function Name	<b><i>void HAL_FIREWALL_GetConfig (FIREWALL_InitTypeDef * fw_config)</i></b>
Function Description	Retrieve the Firewall configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b><i>fw_config:</i></b> Firewall configuration, type is same as</li> </ul>

	initialization structure
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API can't be executed inside a code area protected by the Firewall when the Firewall is enabled</li> <li>• If NVDSL register is different from 0, that is, if the non volatile data segment is defined, this API can't be executed when the Firewall is enabled.</li> <li>• User should resort to __HAL_FIREWALL_GET_PREARM() macro to retrieve FPA bit status</li> </ul>

### HAL\_FIREWALL\_EnableFirewall

Function Name	<b>void HAL_FIREWALL_EnableFirewall (void )</b>
Function Description	Enable FIREWALL.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Firewall is enabled in clearing FWDIS bit of SYSCFG CFGR1 register. Once enabled, the Firewall cannot be disabled by software. Only a system reset can set again FWDIS bit.</li> </ul>

### HAL\_FIREWALL\_EnablePreArmFlag

Function Name	<b>void HAL_FIREWALL_EnablePreArmFlag (void )</b>
Function Description	Enable FIREWALL pre arm.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When FPA bit is set, any code executed outside the protected segment will close the Firewall.</li> <li>• This API provides the same service as __HAL_FIREWALL_PREARM_ENABLE() macro but can't be executed inside a code area protected by the Firewall.</li> <li>• When the Firewall is disabled, user can resort to HAL_FIREWALL_EnablePreArmFlag() API any time.</li> <li>• When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to __HAL_FIREWALL_PREARM_ENABLE() macro instead</li> <li>• When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to __HAL_FIREWALL_PREARM_ENABLE() macro instead.</li> </ul>

### HAL\_FIREWALL\_DisablePreArmFlag

Function Name	<b>void HAL_FIREWALL_DisablePreArmFlag (void )</b>
Function Description	Disable FIREWALL pre arm.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When FPA bit is reset, any code executed outside the</li> </ul>

protected segment when the Firewall is opened will generate a system reset.

- This API provides the same service as `__HAL_FIREWALL_PREARM_DISABLE()` macro but can't be executed inside a code area protected by the Firewall.
- When the Firewall is disabled, user can resort to `HAL_FIREWALL_EnablePreArmFlag()` API any time.
- When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined),  
\*\* this API can be executed when the Firewall is closed \*\*  
when the Firewall is opened, user should resort to  
`__HAL_FIREWALL_PREARM_DISABLE()` macro instead
- When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined)  
\*\* FW\_CR register can be accessed only when the Firewall is opened: user should resort to  
`__HAL_FIREWALL_PREARM_DISABLE()` macro instead.

## 18.3 FIREWALL Firmware driver defines

### 18.3.1 FIREWALL

#### *FIREWALL Exported Macros*

`__HAL_FIREWALL_IS_ENABLED`

#### Description:

- Check whether the FIREWALL is enabled or not.

#### Return value:

- FIREWALL: enabling status (TRUE or FALSE).

`__HAL_FIREWALL_PREARM_ENABLE`

#### Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise it generates a system reset. This macro provides the same service as `HAL_FIREWALL_EnablePreArmFlag()` API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

\_HAL\_FIREWALL\_PREAMP\_DISABLE**Notes:**

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise, it generates a system reset. This macro provides the same service as HAL\_FIREWALL\_DisablePreArmFlag() API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

\_HAL\_FIREWALL\_VOLATILEDATA\_SHARED\_ENABLE**Notes:**

- When VDS bit is set, the volatile data segment is shared with non-protected application code. It can be accessed whatever the Firewall state (opened or closed). This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

\_HAL\_FIREWALL\_VOLATILEDATA\_SHARED\_DISABLE**Notes:**

- When VDS bit is reset, the volatile data segment is not shared and cannot be hit by a non protected executable code when the Firewall is closed. If it is accessed in such a condition, a system reset is generated by the Firewall. This macro can be executed inside a code area

protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

#### \_\_HAL\_FIREWALL\_VOLATILEDATA\_EXECUTION\_ENABLE

##### Notes:

- VDE bit is ignored when VDS is set. If VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is set (with VDS = 0), the volatile data segment is executable. When the Firewall call is closed, a "call gate" entry procedure is required to open first the Firewall. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

#### \_\_HAL\_FIREWALL\_VOLATILEDATA\_EXECUTION\_DISABLE

##### Notes:

- VDE bit is ignored when VDS is set. If VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is reset (with VDS = 0), the volatile data segment cannot be executed. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL

register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

#### \_HAL\_FIREWALL\_GET\_VOLATILEDATA\_SHARED

##### **Description:**

- Check whether or not the volatile data segment is shared.

##### **Return value:**

- VDS: bit setting status (TRUE or FALSE).

##### **Notes:**

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

#### \_HAL\_FIREWALL\_GET\_VOLATILEDATA\_EXECUTI ON

##### **Description:**

- Check whether or not the volatile data segment is declared executable.

##### **Return value:**

- VDE: bit setting status (TRUE or FALSE).

##### **Notes:**

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

#### \_HAL\_FIREWALL\_GET\_PREARM

##### **Description:**

- Check whether or not the Firewall pre arm bit is set.

**Return value:**

- FPA: bit setting status (TRUE or FALSE).

**Notes:**

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

***FIREWALL pre arm status***

`FIREWALL_PRE_ARM_RESET`

`FIREWALL_PRE_ARM_SET`

***FIREWALL volatile data segment execution status***

`FIREWALL_VOLATILEDATA_NOT_EXECUTABLE`

`FIREWALL_VOLATILEDATA_EXECUTABLE`

***FIREWALL volatile data segment share status***

`FIREWALL_VOLATILEDATA_NOT_SHARED`

`FIREWALL_VOLATILEDATA_SHARED`

## 19 HAL FLASH Generic Driver

### 19.1 FLASH Firmware driver registers structures

#### 19.1.1 FLASH\_EraseInitTypeDef

##### Data Fields

- *uint32\_t TypeErase*
- *uint32\_t PageAddress*
- *uint32\_t NbPages*

##### Field Documentation

- *uint32\_t FLASH\_EraseInitTypeDef::TypeErase*  
TypeErase: Page Erase only. This parameter can be a value of  
**FLASHEx\_Type\_Erase**
- *uint32\_t FLASH\_EraseInitTypeDef::PageAddress*  
PageAddress : Initial FLASH address to be erased This parameter must be a value belonging to FLASH Programm address (depending on the devices)
- *uint32\_t FLASH\_EraseInitTypeDef::NbPages*  
NbPages: Number of pages to be erased. This parameter must be a value between 1 and (max number of pages - value of Initial page)

#### 19.1.2 FLASH\_ProcTypeDef

##### Data Fields

- *\_IO FLASH\_ProcTypeDef ProcedureOnGoing*
- *\_IO uint32\_t NbPagesToErase*
- *\_IO uint32\_t Page*
- *\_IO uint32\_t Address*
- *HAL\_LockTypeDef Lock*
- *\_IO uint32\_t ErrorCode*

##### Field Documentation

- *\_IO FLASH\_ProcTypeDef FLASH\_ProcTypeDef::ProcedureOnGoing*
- *\_IO uint32\_t FLASH\_ProcTypeDef::NbPagesToErase*
- *\_IO uint32\_t FLASH\_ProcTypeDef::Page*
- *\_IO uint32\_t FLASH\_ProcTypeDef::Address*
- *HAL\_LockTypeDef FLASH\_ProcTypeDef::Lock*
- *\_IO uint32\_t FLASH\_ProcTypeDef::ErrorCode*

## 19.2 FLASH Firmware driver API description

### 19.2.1 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

### 19.2.2 How to use this driver

This driver provides functions to configure and program the Flash memory of all STM32L0xx devices.

1. FLASH Memory 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.
  - Program functions: Fast Word and Half Page(should be executed from internal SRAM).
2. DATA EEPROM Programming functions: this group includes all needed functions to erase and program the DATA EEPROM memory:
  - Lock and Unlock the DATA EEPROM interface.
  - Erase function: Erase Byte, erase HalfWord, erase Word, erase Double Word (should be executed from internal SRAM).
  - Program functions: Fast Program Byte, Fast Program Half-Word, FastProgramWord, Program Byte, Program Half-Word, Program Word and Program Double-Word (should be executed from internal SRAM).
3. FLASH Option Bytes Programming functions: this group includes all needed functions to:
  - Lock and Unlock the Flash Option bytes.
  - Set/Reset the write protection.
  - Set the Read protection Level.
  - Set the BOR level.
  - Program the user option Bytes.
  - Launch the Option Bytes loader.
  - Get the Write protection.
  - Get the read protection status.
  - Get the BOR level.
  - Get the user option bytes.
4. Interrupts and flags management functions :
  - Handle FLASH interrupts by calling HAL\_FLASH\_IRQHandler()
  - Wait for last FLASH operation according to its status
  - Get error flag status by calling HAL\_GetErrorCode()
5. FLASH Interface configuration functions: this group includes the management of following features:

- Enable/Disable the RUN PowerDown mode.
  - Enable/Disable the SLEEP PowerDown mode.
6. FLASH Peripheral State methods: this group includes the management of following features:
- Wait for the FLASH operation
  - Get the specific FLASH error flag

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 preread buffer
- Enable/Disable the Flash power-down
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

### 19.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

The FLASH Memory Programming functions, includes the following functions:

- HAL\_FLASH\_Unlock(void);
- HAL\_FLASH\_Lock(void);
- HAL\_FLASH\_Program(uint32\_t TypeProgram, uint32\_t Address, uint32\_t Data)
- HAL\_FLASH\_Program\_IT(uint32\_t TypeProgram, uint32\_t Address, uint32\_t Data)

Any operation of erase or program should follow these steps:

1. Call the HAL\_FLASH\_Unlock() function to enable the flash control register and program memory access.
2. Call the desired function to erase page or program data.
3. Call the HAL\_FLASH\_Lock() to disable the flash program memory access (recommended to protect the FLASH memory against possible unwanted operation).

### 19.2.4 Option Bytes Programming functions

The FLASH\_Option Bytes Programming\_functions, includes the following functions:

- HAL\_FLASH\_OB\_Unlock(void);
- HAL\_FLASH\_OB\_Lock(void);
- HAL\_FLASH\_OB\_Launch(void);
- HAL\_FLASHEx\_OBProgram(FLASH\_OBProgramInitTypeDef \*pOBInit);
- HAL\_FLASHEx\_OBGetConfig(FLASH\_OBProgramInitTypeDef \*pOBInit);

Any operation of erase or program should follow these steps:

1. Call the HAL\_FLASH\_OB\_Unlock() function to enable the Flash option control register access.
2. Call the following functions to program the desired option bytes.
  - HAL\_FLASHEx\_OBProgram(FLASH\_OBProgramInitTypeDef \*pOBInit);
3. Once all needed option bytes to be programmed are correctly written, call the HAL\_FLASH\_OB\_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL\_FLASH\_OB\_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection. As a result, these 2 options are exclusive each other.
2. To activate PCROP mode for Flash sectors(s), you need to follow the sequence below:
  - Use this function HAL\_FLASHEx\_AdvOBProgram with PCROPState = OB\_PCROP\_STATE\_ENABLE. \*

### 19.2.5 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\(\)\*\*](#)

### 19.2.6 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\(\)\*\*](#)

### 19.2.7 Detailed description of functions

#### HAL\_FLASH\_Program

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)</b>
Function Description	Program word at a specified address.
Parameters	<ul style="list-style-type: none"> <li>• <b>TypeProgram:</b> Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program</li> <li>• <b>Address:</b> specifies the address to be programmed.</li> <li>• <b>Data:</b> specifies the data to be programmed</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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).</li> </ul>

#### HAL\_FLASH\_Program\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint32_t Data)</b>
Function Description	Program word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TypeProgram:</b> Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program</li> </ul>

- **Address:** specifies the address to be programmed.
  - **Data:** specifies the data to be programmed
- Return values
- **HAL\_StatusTypeDef:** HAL Status

### **HAL\_FLASH\_IRQHandler**

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

### **HAL\_FLASH\_EndOfOperationCallback**

Function Name	<b>void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)</b>
Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>ReturnValue:</b> The value saved in this parameter depends on the ongoing procedure           <ul style="list-style-type: none"> <li>– Pages Erase: Sector which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased)</li> <li>– Program: Address which was selected for data program</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>none:</b></li> </ul>

### **HAL\_FLASH\_OperationErrorHandler**

Function Name	<b>void HAL_FLASH_OperationErrorHandler (uint32_t ReturnValue)</b>
Function Description	FLASH operation error interrupt callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>ReturnValue:</b> The value saved in this parameter depends on the ongoing procedure           <ul style="list-style-type: none"> <li>– Pages Erase: Sector number which returned an error</li> <li>– Program: Address which was selected for data program</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>none:</b></li> </ul>

### **HAL\_FLASH\_Unlock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Unlock (void )</b>
Function Description	Unlock the FLASH control register access.
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>

### **HAL\_FLASH\_Lock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Lock (void )</b>
Function Description	Locks the FLASH control register access.
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>

**HAL\_FLASH\_OB\_Unlock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void )</b>
Function Description	Unlock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"><li>• <b>HAL_StatusTypeDef:</b> HAL Status</li></ul>

**HAL\_FLASH\_OB\_Lock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Lock (void )</b>
Function Description	Lock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"><li>• <b>HAL_StatusTypeDef:</b> HAL Status</li></ul>

**HAL\_FLASH\_OB\_Launch**

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

**HAL\_FLASH\_GetError**

Function Name	<b>uint32_t HAL_FLASH_GetError (void )</b>
Function Description	Get the specific FLASH error flag.
Return values	<ul style="list-style-type: none"><li>• <b>uint32_t:</b> The returned value can be a mixed of :<ul style="list-style-type: none"><li>- HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP)</li><li>- HAL_FLASH_ERROR_SIZE: FLASH Programming Parallelism error flag</li><li>- HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag</li><li>- HAL_FLASH_ERROR_WRP: FLASH Write protected error flag</li><li>- HAL_FLASH_ERROR_OPTV: FLASH Option valid error flag</li><li>- HAL_FLASH_ERROR_FWWERR: FLASH Write or Erase operation aborted</li><li>- HAL_FLASH_ERROR_NOTZERO: FLASH Write operation is done in a not-erased region</li></ul></li></ul>

**FLASH\_WaitForLastOperation**

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

**FLASH\_ErasePage**

Function Name	<b>void FLASH_ErasePage (uint32_t Page_Address)</b>
Function Description	Erases a specified page in program memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>Page_Address:</b> The page address in program memory to be erased.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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)</li> <li>• A Page is erased in the Program memory only if the address to load is the start address of a page (multiple of 128 bytes).</li> </ul>

## 19.3 FLASH Firmware driver defines

### 19.3.1 FLASH

***Flash Error Code***

HAL\_FLASH\_ERROR\_NONE  
 HAL\_FLASH\_ERROR\_RD  
 HAL\_FLASH\_ERROR\_SIZE  
 HAL\_FLASH\_ERROR\_PGA  
 HAL\_FLASH\_ERROR\_WRP  
 HAL\_FLASH\_ERROR\_OPTV  
 HAL\_FLASH\_ERROR\_FWWERR  
 HAL\_FLASH\_ERROR\_NOTZERO

***FLASH Public Constants***

FLASH\_SIZE  
 FLASH\_PAGE\_SIZE

***FLASH Flags***

FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_EOP	FLASH End of Programming flag
FLASH_FLAG_ENDHV	FLASH End of High Voltage flag
FLASH_FLAG_READY	FLASH Ready flag after low power mode
FLASH_FLAG_WRPERR	FLASH Write protected error flag
FLASH_FLAG_PGAERR	FLASH Programming Alignment error flag
FLASH_FLAG_SIZERR	FLASH Size error flag
FLASH_FLAG_OPTVERR	FLASH Option Validity error flag (not valid with STM32L031xx/STM32L041xx)
FLASH_FLAG_RDERR	FLASH Read protected error flag

**FLASH\_FLAG\_FWWERR**      FLASH Write or Erase operation aborted

**FLASH\_FLAG\_NOTZEROERR**    FLASH Read protected error flag

#### ***FLASH Interrupts***

**FLASH\_IT\_EOP**      End of programming interrupt source

**FLASH\_IT\_ERR**      Error interrupt source

#### ***FLASH Keys***

**FLASH\_PDKEY1**      Flash power down key1

**FLASH\_PDKEY2**      Flash power down key2: used with **FLASH\_PDKEY1** to unlock the **RUN\_PD** bit in **FLASH\_ACR**

**FLASH\_PEKEY1**      Flash program erase key1

**FLASH\_PEKEY2**      Flash program erase key: used with **FLASH\_PEKEY1** to unlock the write access to the **FLASH\_PECR** register and data EEPROM

**FLASH\_PRGKEY1**      Flash program memory key1

**FLASH\_PRGKEY2**      Flash program memory key2: used with **FLASH\_PRGKEY1** to unlock the program memory

**FLASH\_OPTKEY1**      Flash option key1

**FLASH\_OPTKEY2**      Flash option key2: used with **FLASH\_OPTKEY1** to unlock the write access to the option byte block

#### ***FLASH Latency***

**FLASH\_LATENCY\_0**      FLASH Zero Latency cycle

**FLASH\_LATENCY\_1**      FLASH One Latency cycle

#### ***Macros to handle FLASH interrupts***

**\_HAL\_FLASH\_ENABLE\_IT**      **Description:**

- Enable the specified FLASH interrupt.

#### **Parameters:**

- **\_INTERRUPT\_**: : FLASH interrupt This parameter can be any combination of the following values:
  - **FLASH\_IT\_EOP**: End of FLASH Operation Interrupt
  - **FLASH\_IT\_ERR**: Error Interrupt

#### **Return value:**

- none

**\_HAL\_FLASH\_DISABLE\_IT**      **Description:**

- Disable the specified FLASH interrupt.

#### **Parameters:**

- **\_INTERRUPT\_**: : FLASH interrupt This parameter can be any combination of the following values:
  - **FLASH\_IT\_EOP**: End of FLASH Operation Interrupt
  - **FLASH\_IT\_ERR**: Error Interrupt

**Return value:**

- none

**\_HAL\_FLASH\_GET\_FLAG**

- 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\_BSY : FLASH Busy flag
  - FLASH\_FLAG\_EOP: FLASH End of Operation flag
  - FLASH\_FLAG\_READY: FLASH Ready flag after low power mode
  - FLASH\_FLAG\_ENDHV: FLASH End of high voltage flag
  - FLASH\_FLAG\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGAERR: FLASH Programming Alignment error flag (not valid with STM32L031xx/STM32L041xx)
  - FLASH\_FLAG\_SIZERR: FLASH Size error flag
  - FLASH\_FLAG\_OPTVERR: FLASH Option validity error flag (not valid with STM32L031xx/STM32L041xx)
  - FLASH\_FLAG\_RDERR: FLASH Read protected error flag
  - FLASH\_FLAG\_FWWERR: FLASH Fetch While Write Error flag
  - FLASH\_FLAG\_NOTZEROERR: Not Zero area error flag

**Return value:**

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

**\_HAL\_FLASH\_CLEAR\_FLAG****Description:**

- Clear the specified FLASH flag.

**Parameters:**

- \_\_FLAG\_\_: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  - FLASH\_FLAG\_EOP: FLASH End of Operation flag
  - FLASH\_FLAG\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGAERR: FLASH Programming Alignment error flag (not valid with STM32L031xx/STM32L041xx)
  - FLASH\_FLAG\_SIZERR: FLASH size error flag
  - FLASH\_FLAG\_OPTVERR: FLASH Option validity error flag (not valid with STM32L031xx/STM32L041xx)

- FLASH\_FLAG\_RDERR: FLASH Read protected error flag
- FLASH\_FLAG\_FWWERR: FLASH Fetch While Write Error flag
- FLASH\_FLAG\_NOTZEROERR: Not Zero area error flag

**Return value:**

- None

***FLASH Type Program***

**FLASH\_TYPEPROGRAM\_WORD** Program a word (32-bit) at a specified address.

## 20 HAL FLASH Extension Driver

### 20.1 FLASHEx Firmware driver registers structures

#### 20.1.1 FLASH\_OBProgramInitTypeDef

##### Data Fields

- *uint32\_t OptionType*
- *uint32\_t WRPState*
- *uint32\_t WRPSector*
- *uint32\_t WRPSector2*
- *uint8\_t RDPLevel*
- *uint8\_t BORLevel*
- *uint8\_t USERConfig*
- *uint8\_t BOOTBit1Config*

##### Field Documentation

- *uint32\_t FLASH\_OBProgramInitTypeDef::OptionType*  
OptionType: Option byte to be configured. This parameter can be a value of [\*FLASHEx\\_Option\\_Type\*](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPState*  
WRPState: Write protection activation or deactivation. This parameter can be a value of [\*FLASHEx\\_WRP\\_State\*](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPSector*  
WRPSector: This bitfield specifies the sector (s) which are write protected. This parameter can be a combination of [\*FLASHEx\\_Option\\_Bytes\\_Write\\_Protection\*](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPSector2*  
WRPSector2 : This bitfield specifies the sector(s) upper Sector31 which are write protected. This parameter can be a combination of [\*FLASHEx\\_Option\\_Bytes\\_Write\\_Protection2\*](#)
- *uint8\_t FLASH\_OBProgramInitTypeDef::RDPLevel*  
RDPLevel: Set the read protection level. This parameter can be a value of [\*FLASHEx\\_Option\\_Bytes\\_Read\\_Protection\*](#)
- *uint8\_t FLASH\_OBProgramInitTypeDef::BORLevel*  
BORLevel: Set the BOR Level. This parameter can be a value of [\*FLASHEx\\_Option\\_Bytes\\_BOR\\_Level\*](#)
- *uint8\_t FLASH\_OBProgramInitTypeDef::USERConfig*  
USERConfig: Program the FLASH User Option Byte: IWDG\_SW / RST\_STOP / RST\_STDBY. This parameter can be a combination of [\*FLASHEx\\_Option\\_Bytes\\_IWatchdog\*](#), [\*FLASHEx\\_Option\\_Bytes\\_nRST\\_STOP\*](#) and [\*FLASHEx\\_Option\\_Bytes\\_nRST\\_STDBY\*](#)
- *uint8\_t FLASH\_OBProgramInitTypeDef::BOOTBit1Config*  
BOOT1Config: Together with input pad Boot0, this bit selects the boot source, flash, ram or system memory This parameter can be a value of [\*FLASHEx\\_Option\\_Bytes\\_BOOTBit1\*](#)

## 20.1.2 **FLASH\_AdvOBProgramInitTypeDef**

### Data Fields

- *uint32\_t OptionType*
- *uint8\_t PCROPState*
- *uint32\_t PCROPSector*
- *uint32\_t PCROPSector2*
- *uint8\_t BootConfig*

### Field Documentation

- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::OptionType***  
OptionType: Option byte to be configured for extension . This parameter can be a value of [\*\*FLASHEx\\_OptionAdv\\_Type\*\*](#)
- ***uint8\_t FLASH\_AdvOBProgramInitTypeDef::PCROPState***  
PCROPState: PCROP activation or deactivation. This parameter can be a value of [\*\*FLASHEx\\_PCROP\\_State\*\*](#)
- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::PCROPSector***  
PCROPSector : This bitfield specifies the sector(s) which are read/write protected. This parameter can be a combination of [\*\*FLASHEx\\_Option\\_Bytes\\_PC\\_ReadWrite\\_Protection\*\*](#)
- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::PCROPSector2***  
PCROPSector : This bitfield specifies the sector(s) upper Sector31 which are read/write protected. This parameter can be a combination of [\*\*FLASHEx\\_Option\\_Bytes\\_PC\\_ReadWrite\\_Protection2\*\*](#)
- ***uint8\_t FLASH\_AdvOBProgramInitTypeDef::BootConfig***  
BootConfig: specifies Option bytes for boot config. This parameter can be a value of [\*\*FLASHEx\\_Option\\_Bytes\\_BOOT\\_BANK\*\*](#)

## 20.2 **FLASHEx Firmware driver API description**

### 20.2.1 **Flash peripheral Extended features**

Comparing to other products, the FLASH interface for STM32L0xx devices contains the following additional features

- Erase functions
- DATA EEPROM memory management
- BOOT option bit configuration
- PCROP protection for all sectors

### 20.2.2 **How to use this driver**

This driver provides functions to configure and program the FLASH memory of all STM32L0xx. It includes:

- Full DATA EEPROM erase and program management
- Boot activation
- PCROP protection configuration and control for all sectors

### 20.2.3 FLASH Erasing Programming functions

The FLASH Memory Erasing functions, includes the following functions:

- HAL\_FLASHEx\_Erase: return only when erase has been done
- HAL\_FLASHEx\_Erase\_IT: end of erase is done when HAL\_FLASH\_EndOfOperationCallback is called with parameter 0xFFFFFFFF

Any operation of erase should follow these steps:

1. Call the 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 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\(\)\*](#)

### 20.2.4 Option Bytes Programming functions

Any operation of erase or program should follow these steps:

1. Call the HAL\_FLASH\_OB\_Unlock() function to enable the Flash option control register access.
2. Call following function to program the desired option bytes.
  - HAL\_FLASHEx\_OBProgram: - To Enable/Disable the desired sector write protection. - To set the desired read Protection Level. - To configure the user option Bytes: IWDG, STOP and the Standby. - To Set the BOR level.
3. Once all needed option bytes to be programmed are correctly written, call the HAL\_FLASH\_OB\_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL\_FLASH\_OB\_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection (nWRPi bits). As a result, these 2 options are exclusive each other.
2. In order to activate the PcROP (change the function of the nWRPi option bits), the WPRMOD option bit must be activated.
3. The active value of nWRPi bits is inverted when PCROP mode is active, this means: if WPRMOD = 1 and nWRPi = 1 (default value), then the user sector "i" is read/write protected.
4. To activate PCROP mode for Flash sector(s), you need to call the following function:
  - HAL\_FLASHEx\_AdvOBProgram in selecting sectors to be read/write protected
  - HAL\_FLASHEx\_OB\_SelectPCROP to enable the read/write protection

This section contains the following APIs:

- [\*HAL\\_FLASHEx\\_OBProgram\(\)\*](#)
- [\*HAL\\_FLASHEx\\_OBGetConfig\(\)\*](#)
- [\*HAL\\_FLASHEx\\_AdvOBProgram\(\)\*](#)
- [\*HAL\\_FLASHEx\\_AdvOBGetConfig\(\)\*](#)
- [\*HAL\\_FLASHEx\\_OB\\_SelectPCROP\(\)\*](#)
- [\*HAL\\_FLASHEx\\_OB\\_DeSelectPCROP\(\)\*](#)

## 20.2.5 DATA EEPROM Programming functions

The FLASH\_DATAEEPROM Programming Functions, includes the following functions:

- HAL\_FLASHEx\_DATAEEPROM\_Unlock(void);
- HAL\_FLASHEx\_DATAEEPROM\_Lock(void);
- HAL\_FLASHEx\_DATAEEPROM\_Erase(uint32\_t Address)
- HAL\_FLASHEx\_DATAEEPROM\_Program(uint32\_t TypeProgram, uint32\_t Address, uint32\_t Data)

Any operation of erase or program should follow these steps:

1. Call the HAL\_FLASHEx\_DATAEEPROM\_Unlock() function to enable the data EEPROM access and Flash program erase control register access.
2. Call the desired function to erase or program data.
3. Call the HAL\_FLASHEx\_DATAEEPROM\_Lock() to disable the data EEPROM access and Flash program erase control register access(recommended to protect the DATA EEPROM against possible unwanted operation).

This section contains the following APIs:

- [\*\*HAL\\_FLASHEx\\_DATAEEPROM\\_Unlock\(\)\*\*](#)
- [\*\*HAL\\_FLASHEx\\_DATAEEPROM\\_Lock\(\)\*\*](#)
- [\*\*HAL\\_FLASHEx\\_DATAEEPROM\\_Erase\(\)\*\*](#)
- [\*\*HAL\\_FLASHEx\\_DATAEEPROM\\_Program\(\)\*\*](#)
- [\*\*HAL\\_FLASHEx\\_DATAEEPROM\\_EnableFixedTimeProgram\(\)\*\*](#)
- [\*\*HAL\\_FLASHEx\\_DATAEEPROM\\_DisableFixedTimeProgram\(\)\*\*](#)

## 20.2.6 Detailed description of functions

### **HAL\_FLASHEx\_Erase**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)</b>
Function Description	Erase the specified FLASH memory Pages.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit:</b> pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.</li> <li>• <b>PageError:</b> pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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)</li> </ul>

### **HAL\_FLASHEx\_Erase\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)</b>
Function Description	Perform a page erase of the specified FLASH memory pages with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit:</b> pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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).</li> <li>End of erase is done when HAL_FLASH_EndOfOperationCallback is called with parameter 0xFFFFFFFF</li> </ul>

### HAL\_FLASHEx\_OBProgram

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

### HAL\_FLASHEx\_OBGetConfig

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

### HAL\_FLASHEx\_AdvOBProgram

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)</b>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> <li>• <b>pAdvOBInit:</b> pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>

### HAL\_FLASHEx\_AdvOBGetConfig

Function Name	<b>void HAL_FLASHEx_AdvOBGetConfig (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)</b>
Function Description	Get the OBEX byte configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>pAdvOBInit:</b> pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_FLASHEx\_OB\_SelectPCROP**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP (void )</b>
Function Description	Select the Protection Mode (WPRMOD).
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once WPRMOD bit is active, unprotection of a protected sector is not possible</li> <li>• Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag</li> </ul>

**HAL\_FLASHEx\_OB\_DeSelectPCROP**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP (void )</b>
Function Description	Deselect the Protection Mode (WPRMOD).
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once WPRMOD bit is active, unprotection of a protected sector is not possible</li> <li>• Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag</li> </ul>

**HAL\_FLASHEx\_DATAEEPROM\_Unlock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Unlock (void )</b>
Function Description	Unlocks the data memory and FLASH_PECR register access.
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>

**HAL\_FLASHEx\_DATAEEPROM\_Lock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Lock (void )</b>
Function Description	Locks the Data memory and FLASH_PECR register access.
Return values	<ul style="list-style-type: none"> <li>• <b>HAL_StatusTypeDef:</b> HAL Status</li> </ul>

**HAL\_FLASHEx\_DATAEEPROM\_Erase**

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Erase (uint32_t Address)</b>
Function Description	Erase a word in data memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>Address:</b> specifies the address to be erased.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To correctly run this function, the HAL_FLASHEx_DATAEEPROM_Unlock() function must be called before. Call the HAL_FLASHEx_DATAEEPROM_Lock() to the data EEPROM access and Flash program erase control register</li> </ul>

access(recommended to protect the DATA\_EEPROM against possible unwanted operation).

### **HAL\_FLASHEx\_DATAEEPROM\_Program**

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

### **HAL\_FLASHEx\_DATAEEPROM\_EnableFixedTimeProgram**

Function Name	<b>void HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram (void )</b>
Function Description	Enable DATA EEPROM fixed Time programming (2*Tprog).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_FLASHEx\_DATAEEPROM\_DisableFixedTimeProgram**

Function Name	<b>void HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram (void )</b>
Function Description	Disables DATA EEPROM fixed Time programming (2*Tprog).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## **20.3 FLASHEx Firmware driver defines**

### **20.3.1 FLASHEx**

#### ***FLASHEx Address***

#### ***FLASH\_NBPAGES\_MAX***

#### ***FLASHEx Exported Macros***

***\_HAL\_FLASH\_SET\_LATENCY***

#### **Description:**

- Set the FLASH Latency.

#### **Parameters:**

- **\_LATENCY\_**: FLASH Latency This parameter can be one of the following values:
  - **FLASH\_LATENCY\_0:** FLASH Zero Latency cycle
  - **FLASH\_LATENCY\_1:**

FLASH One Latency cycle

**Return value:**

- none

`_HAL_FLASH_GET_LATENCY`

**Description:**

- Get the FLASH Latency.

**Return value:**

- FLASH: Latency This parameter can be one of the following values:
  - FLASH\_LATENCY\_0: FLASH Zero Latency cycle
  - FLASH\_LATENCY\_1: FLASH One Latency cycle

`_HAL_FLASH_PREFETCH_BUFFER_ENABLE`

**Description:**

- Enable/Disable the FLASH prefetch buffer.

**Return value:**

- none

`_HAL_FLASH_PREFETCH_BUFFER_DISABLE`

**Description:**

- Enable/Disable the FLASH Buffer cache.

**Return value:**

- none

`_HAL_FLASH_BUFFER_CACHE_DISABLE`

**Description:**

- Enable/Disable the FLASH buffer cache.

**Return value:**

- none

`_HAL_FLASH_PREREAD_BUFFER_DISABLE`

**Description:**

- Enable/Disable the FLASH preread buffer.

**Return value:**

- none

`_HAL_FLASH_SLEEP_POWERDOWN_DISABLE`

**Notes:**

- Writing this bit to 0 this bit,

automatically the keys are lost  
and a new unlock sequence is  
necessary to re-write it to 1.

#### \_HAL\_FLASH\_POWER\_DOWN\_DISABLE

#### Notes:

- Writing this bit to 0 this bit,  
automatically the keys are lost  
and a new unlock sequence is  
necessary to re-write it to 1.

#### ***FLASH Option Byte***

OPTIONBYTE_PCROP	PCROP option byte configuration
OPTIONBYTE_BOOTCONFIG	BOOTConfig option byte configuration, boot from bank 2

#### ***FLASH Option Bytes BOOT Bit1 Setup***

OB_BOOT_BIT1_RESET	BOOT Bit 1 Reset
OB_BOOT_BIT1_SET	BOOT Bit 1 Set

#### ***FLASH Option Bytes BOOT BANK***

OB_BOOT_BANK1	At startup, if boot pin 0 and BOOT1 bit are set in boot from user Flash position and this parameter is selected the device will boot from Bank 1 (Default)
OB_BOOT_BANK2	At startup, if boot pin 0 and BOOT1 bit are set in boot from user Flash position and this parameter is selected the device will boot from Bank 2

#### ***FLASH Option Bytes BOR Level***

OB_BOR_OFF	BOR is disabled at power down, the reset is asserted when the VDD power supply reaches the PDR(Power Down Reset) threshold (1.5V)
OB_BOR_LEVEL1	BOR Reset threshold levels for 1.7V - 1.8V VDD power supply
OB_BOR_LEVEL2	BOR Reset threshold levels for 1.9V - 2.0V VDD power supply
OB_BOR_LEVEL3	BOR Reset threshold levels for 2.3V - 2.4V VDD power supply
OB_BOR_LEVEL4	BOR Reset threshold levels for 2.55V - 2.65V VDD power supply
OB_BOR_LEVEL5	BOR Reset threshold levels for 2.8V - 2.9V VDD power supply

#### ***FLASH Option Bytes IWatchdog***

OB_IWDG_SW	Software WDG selected
OB_IWDG_HW	Hardware WDG selected

#### ***FLASH Option Bytes nRST\_STDBY***

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

#### ***FLASHEx Option Bytes nRST\_STOP***

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

#### ***FLASH Option Bytes PC Read/Write Protection***

OB\_PCROP\_Pages0to31  
OB\_PCROP\_Pages32to63  
OB\_PCROP\_Pages64to95  
OB\_PCROP\_Pages96to127  
OB\_PCROP\_Pages128to159  
OB\_PCROP\_Pages160to191  
OB\_PCROP\_Pages192to223  
OB\_PCROP\_Pages224to255  
OB\_PCROP\_Pages256to287  
OB\_PCROP\_Pages288to319  
OB\_PCROP\_Pages320to351  
OB\_PCROP\_Pages352to383  
OB\_PCROP\_Pages384to415  
OB\_PCROP\_Pages416to447  
OB\_PCROP\_Pages448to479  
OB\_PCROP\_Pages480to511  
OB\_PCROP\_Pages512to543  
OB\_PCROP\_Pages544to575  
OB\_PCROP\_Pages576to607  
OB\_PCROP\_Pages608to639  
OB\_PCROP\_Pages640to671  
OB\_PCROP\_Pages672to703  
OB\_PCROP\_Pages704to735  
OB\_PCROP\_Pages736to767  
OB\_PCROP\_Pages768to799  
OB\_PCROP\_Pages800to831  
OB\_PCROP\_Pages832to863  
OB\_PCROP\_Pages864to895  
OB\_PCROP\_Pages896to927  
OB\_PCROP\_Pages928to959  
OB\_PCROP\_Pages960to991  
OB\_PCROP\_Pages992to1023  
OB\_PCROP\_AllPages            PC Read/Write protection of all Sectors  
**FLASH Option Bytes PC Read/Write Protection (Sector 2)**

OB\_PCROP2\_Pages1024to1055  
OB\_PCROP2\_Pages1056to1087

OB\_PCROP2\_Pages1088to1119  
OB\_PCROP2\_Pages1120to1151  
OB\_PCROP2\_Pages1152to1183  
OB\_PCROP2\_Pages1184to1215  
OB\_PCROP2\_Pages1216to1247  
OB\_PCROP2\_Pages1248to1279  
OB\_PCROP2\_Pages1280to1311  
OB\_PCROP2\_Pages1312to1343  
OB\_PCROP2\_Pages1344to1375  
OB\_PCROP2\_Pages1376to1407  
OB\_PCROP2\_Pages1408to1439  
OB\_PCROP2\_Pages1440to1471  
OB\_PCROP2\_Pages1472to1503  
OB\_PCROP2\_Pages1504to1535  
OB\_PCROP2\_AllPages                    PC Read/Write protection of all Sectors PCROP2

***FLASH Option Bytes Write Mask***

WRP\_MASK\_LOW  
WRP\_MASK\_HIGH

***FLASH Option Bytes Read Protection***

OB\_RDP\_LEVEL\_0  
OB\_RDP\_LEVEL\_1  
OB\_RDP\_LEVEL\_2

***FLASH Option Bytes Write ProtectionP***

OB\_WRP\_Pages0to31  
OB\_WRP\_Pages32to63  
OB\_WRP\_Pages64to95  
OB\_WRP\_Pages96to127  
OB\_WRP\_Pages128to159  
OB\_WRP\_Pages160to191  
OB\_WRP\_Pages192to223  
OB\_WRP\_Pages224to255  
OB\_WRP\_Pages256to287  
OB\_WRP\_Pages288to319  
OB\_WRP\_Pages320to351  
OB\_WRP\_Pages352to383  
OB\_WRP\_Pages384to415

OB\_WRP\_Pages416to447  
OB\_WRP\_Pages448to479  
OB\_WRP\_Pages480to511  
OB\_WRP\_Pages512to543  
OB\_WRP\_Pages544to575  
OB\_WRP\_Pages576to607  
OB\_WRP\_Pages608to639  
OB\_WRP\_Pages640to671  
OB\_WRP\_Pages672to703  
OB\_WRP\_Pages704to735  
OB\_WRP\_Pages736to767  
OB\_WRP\_Pages768to799  
OB\_WRP\_Pages800to831  
OB\_WRP\_Pages832to863  
OB\_WRP\_Pages864to895  
OB\_WRP\_Pages896to927  
OB\_WRP\_Pages928to959  
OB\_WRP\_Pages960to991  
OB\_WRP\_Pages992to1023  
OB\_WRP\_AllPages                          Write protection of all Sectors

***FLASH Option Bytes Write Protection***

OB\_WRP2\_Pages1024to1055  
OB\_WRP2\_Pages1056to1087  
OB\_WRP2\_Pages1088to1119  
OB\_WRP2\_Pages1120to1151  
OB\_WRP2\_Pages1152to1183  
OB\_WRP2\_Pages1184to1215  
OB\_WRP2\_Pages1216to1247  
OB\_WRP2\_Pages1248to1279  
OB\_WRP2\_Pages1280to1311  
OB\_WRP2\_Pages1312to1343  
OB\_WRP2\_Pages1344to1375  
OB\_WRP2\_Pages1376to1407  
OB\_WRP2\_Pages1408to1439  
OB\_WRP2\_Pages1440to1471  
OB\_WRP2\_Pages1472to1503

OB\_WRP2\_Pages1504to1535

OB\_WRP2\_AllPages      Write protection of all Sectors WRP2

***FLASH Option Type***

OPTIONBYTE\_WRP      WRP option byte configuration

OPTIONBYTE\_RDP      RDP option byte configuration

OPTIONBYTE\_USER      USER option byte configuration

OPTIONBYTE\_BOR      BOR option byte configuration

OPTIONBYTE\_BOOT\_BIT1      BOOT PIN1 option byte configuration

***FLASH PCROP State***

OB\_PCROP\_STATE\_DISABLE      Disable PCROP

OB\_PCROP\_STATE\_ENABLE      Enable PCROP

***FLASH Type Erase***

FLASH\_TYPEERASE\_PAGES      Page erase only

***FLASH Type Program Data***

FLASH\_TYPEPROGRAMDATA\_BYTE      Program byte (8-bit) at a specified address.

FLASH\_TYPEPROGRAMDATA\_HALFWORD      Program a half-word (16-bit) at a specified address.

FLASH\_TYPEPROGRAMDATA\_WORD      Program a word (32-bit) at a specified address.

FLASH\_TYPEPROGRAM\_BYTE

FLASH\_TYPEPROGRAM\_HALFWORD

***FLASH WRP State***

OB\_WRPSTATE\_DISABLE      Disable the write protection of the desired sectors

OB\_WRPSTATE\_ENABLE      Enable the write protection of the desired sectors

## 21 HAL FLASH\_\_RAMFUNC Generic Driver

### 21.1 FLASH\_\_RAMFUNC Firmware driver API description

#### 21.1.1 Detailed description of functions

##### HAL\_FLASHEx\_HalfPageProgram

Function Name	<code>__RAM_FUNC HAL_FLASHEx_HalfPageProgram (uint32_t Address, uint32_t * pBuffer)</code>
Function Description	Program a half page in program memory.
Parameters	<ul style="list-style-type: none"><li><b>Address:</b> specifies the address to be written.</li><li><b>pBuffer:</b> pointer to the buffer containing the data to be written to the half page.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL:</b> Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.</li></ul>
Notes	<ul style="list-style-type: none"><li>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)</li><li>Half page write is possible only from SRAM.</li><li>A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 64 bytes) and the 15 remaining words to load are in the same half page.</li><li>During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.).</li><li>If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.</li></ul>

##### HAL\_FLASHEx\_EnableRunPowerDown

Function Name	<code>__RAM_FUNC HAL_FLASHEx_EnableRunPowerDown (void )</code>
Function Description	Enable the power down mode during RUN mode.
Return values	<ul style="list-style-type: none"><li><b>HAL:</b> Status</li></ul>
Notes	<ul style="list-style-type: none"><li>This function can be used only when the user code is running from Internal SRAM.</li></ul>

##### HAL\_FLASHEx\_DisableRunPowerDown

Function Name	<code>__RAM_FUNC HAL_FLASHEx_DisableRunPowerDown (void )</code>
Function Description	Disable the power down mode during RUN mode.

Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function can be used only when the user code is running from Internal SRAM.</li> </ul>

### HAL\_FLASHRAM\_GetError

Function Name	<code>__RAM_FUNC HAL_FLASHRAM_GetError (uint32_t * error)</code>
Function Description	Get the specific FLASH errors flag.
Parameters	<ul style="list-style-type: none"> <li><b>error:</b> pointer is the error value. It can be a mixed of : <ul style="list-style-type: none"> <li>- HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP)</li> <li>- HAL_FLASH_ERROR_SIZE: FLASH Programming Parallelism error flag</li> <li>- HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag</li> <li>- HAL_FLASH_ERROR_WRP: FLASH Write protected error flag</li> <li>- HAL_FLASH_ERROR_OPTV: FLASH Option valid error flag</li> <li>- HAL_FLASH_ERROR_FWWERR: FLASH Write or Erase operation aborted</li> <li>- HAL_FLASH_ERROR_NOTZERO: FLASH Write operation is done in a not-erased region</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> Status</li> </ul>

### HAL\_FLASHEx\_EraseParallelPage

Function Name	<code>__RAM_FUNC HAL_FLASHEx_EraseParallelPage (uint32_t Page_Address1, uint32_t Page_Address2)</code>
Function Description	Erases a specified 2 pages in program memory in parallel.
Parameters	<ul style="list-style-type: none"> <li><b>Page_Address1:</b> The page address in program memory to be erased in the first Bank (BANK1). This parameter should be between FLASH_BASE and FLASH_BANK1_END.</li> <li><b>Page_Address2:</b> The page address in program memory to be erased in the second Bank (BANK2). This parameter should be between FLASH_BANK2_BASE and FLASH_BANK2_END.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function can be used only for STM32L07xxx/STM32L08xxx devices. 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).</li> <li>A Page is erased in the Program memory only if the address to load is the start address of a page (multiple of 128 bytes).</li> </ul>

**HAL\_FLASHEx\_ProgramParallelHalfPage**

Function Name	<code>__RAM_FUNC HAL_FLASHEx_ProgramParallelHalfPage (uint32_t Address1, uint32_t * pBuffer1, uint32_t Address2, uint32_t * pBuffer2)</code>
Function Description	Programs 2 half pages in program memory in parallel.
Parameters	<ul style="list-style-type: none"><li>• <b>Address1:</b> specifies the first address to be written in the first bank (BANK1). This parameter should be between FLASH_BASE and (FLASH_BANK1_END - FLASH_PAGE_SIZE).</li><li>• <b>pBuffer1:</b> pointer to the buffer containing the data to be written to the first half page in the first bank.</li><li>• <b>Address2:</b> specifies the second address to be written in the second bank (BANK2). This parameter should be between FLASH_BANK2_BASE and (FLASH_BANK2_END - FLASH_PAGE_SIZE).</li><li>• <b>pBuffer2:</b> pointer to the buffer containing the data to be written to the second half page in the second bank.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.</li></ul>
Notes	<ul style="list-style-type: none"><li>• This function can be used only for STM32L07xxx/STM32L08xxx devices.</li><li>• 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).</li><li>• Half page write is possible only from SRAM.</li><li>• A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 64 bytes) and the 15 remaining words to load are in the same half page.</li><li>• During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.).</li><li>• If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.</li></ul>

## 22 HAL GPIO Generic Driver

### 22.1 GPIO Firmware driver registers structures

#### 22.1.1 GPIO\_InitTypeDef

##### Data Fields

- *uint32\_t Pin*
- *uint32\_t Mode*
- *uint32\_t Pull*
- *uint32\_t Speed*
- *uint32\_t Alternate*

##### Field Documentation

- ***uint32\_t GPIO\_InitTypeDef::Pin***  
Specifies the GPIO pins to be configured. This parameter can be a combination of [\*\*GPIO\\_pins\\_define\*\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Mode***  
Specifies the operating mode for the selected pins. This parameter can be a value of [\*\*GPIO\\_mode\\_define\*\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Pull***  
Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of [\*\*GPIO\\_pull\\_define\*\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Speed***  
Specifies the speed for the selected pins. This parameter can be a value of [\*\*GPIO\\_speed\\_define\*\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Alternate***  
Peripheral to be connected to the selected pins This parameter can be a value of [\*\*GPIOEx\\_Alternate\\_function\\_selection\*\*](#)

## 22.2 GPIO Firmware driver API description

### 22.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.

- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.
- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.
- The external interrupt/event controller consists of up to 28 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

## 22.2.2 How to use this driver

1. Enable the GPIO IOPORT clock using the following function:  
`_HAL_RCC_GPIOx_CLK_ENABLE()`.
2. Configure the GPIO pin(s) using `HAL_GPIO_Init()`.
  - Configure the IO mode using "Mode" member from `GPIO_InitTypeDef` structure
  - Activate Pull-up, Pull-down resistor using "Pull" member from `GPIO_InitTypeDef` structure.
  - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from `GPIO_InitTypeDef` structure.
  - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure.
  - Analog mode is required when a pin is to be used as ADC channel or DAC output.
  - In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. `HAL_GPIO_DeInit` allows to set register values to their reset value. This function is also to be used when unconfiguring pin which was used as an external interrupt or in event mode. That is the only way to reset the corresponding bit in EXTI & SYSCFG registers.
5. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
6. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()`/`HAL_GPIO_TogglePin()`.
7. To lock pin configuration until next reset use `HAL_GPIO_LockPin()`.
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. 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.
10. The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

## 22.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- `HAL_GPIO_Init()`
- `HAL_GPIO_DeInit()`

## 22.2.4 IO operation functions

This section contains the following APIs:

- `HAL_GPIO_ReadPin()`
- `HAL_GPIO_WritePin()`
- `HAL_GPIO_TogglePin()`
- `HAL_GPIO_LockPin()`
- `HAL_GPIO_EXTI_IRQHandler()`
- `HAL_GPIO_EXTI_Callback()`

## 22.2.5 Detailed description of functions

### `HAL_GPIO_Init`

Function Name	<code>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..E and H) to select the GPIO peripheral for STM32L0XX family devices. Note that GPIOE is not available on all devices.</li> <li>• <b>GPIO_InitStruct:</b> pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### `HAL_GPIO_DeInit`

Function Name	<code>void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</code>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..E and H) to select the GPIO peripheral for STM32L0XX family devices. Note that GPIOE is not available on all devices.</li> <li>• <b>GPIO_Pin:</b> specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### `HAL_GPIO_ReadPin`

Function Name	<code>GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is</li> </ul>

- not available on all devices.
- **GPIO\_Pin:** specifies the port bit to read. This parameter can be GPIO\_PIN\_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.
  - **The:** input port pin value.

### **HAL\_GPIO\_WritePin**

Function Name	<b>void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</b>
Function Description	Sets or clears the selected data port bit.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices.</li> <li>• <b>GPIO_Pin:</b> specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.</li> <li>• <b>PinState:</b> specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pin GPIO_PIN_SET: to set the port pin</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.</li> </ul>

### **HAL\_GPIO\_TogglePin**

Function Name	<b>void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Toggles the specified GPIO pins.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> Where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices. All port bits are not necessarily available on all GPIOs.</li> <li>• <b>GPIO_Pin:</b> Specifies the pins to be toggled.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_GPIO\_LockPin**

Function Name	<b>HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Locks GPIO Pins configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family. Note that GPIOE is not available on all devices.</li> <li>• <b>GPIO_Pin:</b> specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15). All port bits are not necessarily available on all</li> </ul>

	GPIOs.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.</li> <li>• The configuration of the locked GPIO pins can no longer be modified until the next reset.</li> </ul>

### HAL\_GPIO\_EXTI\_IRQHandler

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

### HAL\_GPIO\_EXTI\_Callback

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

## 22.3 GPIO Firmware driver defines

### 22.3.1 GPIO

#### *GPIO Exported Constants*

GPIO\_PIN\_MASK  
IS\_GPIO\_PIN  
IS\_GPIO\_MODE  
IS\_GPIO\_SPEED  
IS\_GPIO\_PULL

#### *GPIO Exported Macros*

<code>_HAL_GPIO_EXTI_GET_FLAG</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Checks whether the specified EXTI line flag is set or not.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <code>_EXTI_LINE_</code>: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• The new state of <code>_EXTI_LINE_</code> (SET or RESET).</li> </ul>

`__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 Exported Types**`IS_GPIO_PIN_ACTION`**Mode definition**

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

***Pin definition***

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

***Pull definition***

GPIO_NOPULL	No Pull-up or Pull-down activation
-------------	------------------------------------

---

GPIO_PULLUP	Pull-up activation
GPIO_PULLDOWN	Pull-down activation

***Speed definition***

GPIO_SPEED_FREQ_LOW	range up to 0.4 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_MEDIUM	range 0.4 MHz to 2 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_HIGH	range 2 MHz to 10 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ VERY HIGH	range 10 MHz to 35 MHz, please refer to the product datasheet

## 23 HAL GPIO Extension Driver

### 23.1 GPIOEx Firmware driver defines

#### 23.1.1 GPIOEx

##### *Alternate function selection*

```
GPIO_AF0_EVENTOUT  
GPIO_AF0_TIM21  
GPIO_AF0_SPI1  
GPIO_AF0_MCO  
GPIO_AF0_SWDIO  
GPIO_AF0_SWCLK  
GPIO_AF0_USART1  
GPIO_AF0_SPI2  
GPIO_AF0_LPTIM1  
GPIO_AF0_TIM22  
GPIO_AF0_LPUART1  
GPIO_AF0_USART2  
GPIO_AF0_TIM2  
GPIO_AF0_USB  
GPIO_AF1_I2C1  
GPIO_AF1_SPI2  
GPIO_AF1_TIM21  
GPIO_AF1_LCD  
GPIO_AF2_TIM2  
GPIO_AF2_TIM3  
GPIO_AF2_EVENTOUT  
GPIO_AF2_LPTIM1  
GPIO_AF2_LPUART1  
GPIO_AF2_MCO  
GPIO_AF2_RTC  
GPIO_AF2_SPI2  
GPIO_AF2_USART5  
GPIO_AF2_SPI1  
GPIO_AF2_USB  
GPIO_AF3_EVENTOUT
```

GPIO\_AF3\_I2C1  
GPIO\_AF3\_TSC  
GPIO\_AF4\_USART2  
GPIO\_AF4\_LPUART1  
GPIO\_AF4\_USART1  
GPIO\_AF4\_EVENTOUT  
GPIO\_AF4\_TIM22  
GPIO\_AF4\_TIM3  
GPIO\_AF4\_I2C1  
GPIO\_AF5\_TIM2  
GPIO\_AF5\_TIM21  
GPIO\_AF5\_TIM22  
GPIO\_AF5\_USART1  
GPIO\_AF5\_SPI2  
GPIO\_AF5\_I2C2  
GPIO\_AF6\_USART4  
GPIO\_AF6\_LPUART1  
GPIO\_AF6\_EVENTOUT  
GPIO\_AF6\_I2C1  
GPIO\_AF6\_I2C2  
GPIO\_AF6\_USART5  
GPIO\_AF6\_TIM21  
GPIO\_AF7\_COMP1  
GPIO\_AF7\_COMP2  
GPIO\_AF7\_I2C3  
GPIO\_AF7\_LPUART1

***Pin available***

GPIOA\_PIN\_AVAILABLE  
GPIOB\_PIN\_AVAILABLE  
GPIOC\_PIN\_AVAILABLE  
GPIOD\_PIN\_AVAILABLE  
GPIOE\_PIN\_AVAILABLE  
GPIOH\_PIN\_AVAILABLE

## 24 HAL I2C Generic Driver

### 24.1 I2C Firmware driver registers structures

#### 24.1.1 I2C\_InitTypeDef

##### Data Fields

- *uint32\_t Timing*
- *uint32\_t OwnAddress1*
- *uint32\_t AddressingMode*
- *uint32\_t DualAddressMode*
- *uint32\_t OwnAddress2*
- *uint32\_t OwnAddress2Masks*
- *uint32\_t GeneralCallMode*
- *uint32\_t NoStretchMode*

##### Field Documentation

- ***uint32\_t I2C\_InitTypeDef::Timing***  
Specifies the I2C\_TIMINGR\_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32\_t I2C\_InitTypeDef::OwnAddress1***  
Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32\_t I2C\_InitTypeDef::AddressingMode***  
Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [\*I2C\\_addressing\\_mode\*](#)
- ***uint32\_t I2C\_InitTypeDef::DualAddressMode***  
Specifies if dual addressing mode is selected. This parameter can be a value of [\*I2C\\_dual\\_addressing\\_mode\*](#)
- ***uint32\_t I2C\_InitTypeDef::OwnAddress2***  
Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- ***uint32\_t I2C\_InitTypeDef::OwnAddress2Masks***  
Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of [\*I2C\\_own\\_address2\\_masks\*](#)
- ***uint32\_t I2C\_InitTypeDef::GeneralCallMode***  
Specifies if general call mode is selected. This parameter can be a value of [\*I2C\\_general\\_call\\_addressing\\_mode\*](#)
- ***uint32\_t I2C\_InitTypeDef::NoStretchMode***  
Specifies if nostretch mode is selected. This parameter can be a value of [\*I2C\\_nostretch\\_mode\*](#)

#### 24.1.2 I2C\_HandleTypeDef

##### Data Fields



- *I2C\_TypeDef \* Instance*
- *I2C\_InitTypeDef Init*
- *uint8\_t \* pBuffPtr*
- *uint16\_t XferSize*
- *\_IO uint16\_t XferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *\_IO HAL\_I2C\_StateTypeDef State*
- *\_IO uint32\_t ErrorCode*

#### Field Documentation

- *I2C\_TypeDef\* I2C\_HandleTypeDef::Instance*  
I2C registers base address
- *I2C\_InitTypeDef I2C\_HandleTypeDef::Init*  
I2C communication parameters
- *uint8\_t\* I2C\_HandleTypeDef::pBuffPtr*  
Pointer to I2C transfer buffer
- *uint16\_t I2C\_HandleTypeDef::XferSize*  
I2C transfer size
- *\_IO uint16\_t I2C\_HandleTypeDef::XferCount*  
I2C transfer counter
- *DMA\_HandleTypeDef\* I2C\_HandleTypeDef::hdmatx*  
I2C Tx DMA handle parameters
- *DMA\_HandleTypeDef\* I2C\_HandleTypeDef::hdmarx*  
I2C Rx DMA handle parameters
- *HAL\_LockTypeDef I2C\_HandleTypeDef::Lock*  
I2C locking object
- *\_IO HAL\_I2C\_StateTypeDef I2C\_HandleTypeDef::State*  
I2C communication state
- *\_IO uint32\_t I2C\_HandleTypeDef::ErrorCode*  
I2C Error code, see I2C\_Error\_Code

## 24.2 I2C Firmware driver API description

### 24.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C\_HandleTypeDef handle structure, for example: I2C\_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implement the HAL\_I2C\_MspInit ()API:
  - a. Enable the I2Cx interface clock
  - b. I2C pins configuration
    - Enable the clock for the I2C GPIOs
    - Configure I2C pins as alternate function open-drain
  - c. NVIC configuration if you need to use interrupt process
    - Configure the I2Cx interrupt priority
    - Enable the NVIC I2C IRQ Channel
  - d. DMA Configuration if you need to use DMA process

- Declare a DMA\_HandleTypeDef handle structure for the transmit or receive 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 Clock Timing, Own Address1, Master Addressing Mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
  4. Initialize the I2C registers by calling the HAL\_I2C\_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL\_I2C\_MspInit(&hi2c) API.
  5. To check if target device is ready for communication, use the function HAL\_I2C\_IsDeviceReady()
  6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

### **Polling mode IO operation**

- Transmit in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Receive()

### **Polling mode IO MEM operation**

- Write an amount of data in blocking mode to a specific memory address using HAL\_I2C\_Mem\_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL\_I2C\_Mem\_Read()

### **Interrupt mode IO operation**

- Transmit in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Transmit\_IT()
- At transmission end of transfer HAL\_I2C\_MasterTxCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCallback
- 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\_MasterRxCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Transmit\_IT()

- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### Interrupt mode IO MEM operation

- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL\_I2C\_Mem\_Write\_IT()
- At MEM end of write transfer HAL\_I2C\_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemTxCpltCallback
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL\_I2C\_Mem\_Read\_IT()
- At MEM end of read transfer HAL\_I2C\_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### DMA mode IO operation

- Transmit in master mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Master\_Transmit\_DMA()
- At transmission end of transfer HAL\_I2C\_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Master\_Receive\_DMA()
- At reception end of transfer HAL\_I2C\_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Slave\_Transmit\_DMA()
- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Slave\_Receive\_DMA()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL\_I2C\_Mem\_Write\_DMA()

- At MEM end of write transfer HAL\_I2C\_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemTxCpltCallback
- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL\_I2C\_Mem\_Read\_DMA()
- At MEM end of read transfer HAL\_I2C\_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- \_\_HAL\_I2C\_ENABLE: Enable the I2C peripheral
- \_\_HAL\_I2C\_DISABLE: Disable the I2C peripheral
- \_\_HAL\_I2C\_GET\_FLAG : Check 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

#### 24.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Cx peripheral:

- User must Implement HAL\_I2C\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_I2C\_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function HAL\_I2C\_DelInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- [\*\*HAL\\_I2C\\_Init\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_MspDelInit\(\)\*\*](#)

#### 24.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:

- Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
- HAL\_I2C\_Master\_Transmit()
  - HAL\_I2C\_Master\_Receive()
  - HAL\_I2C\_Slave\_Transmit()
  - HAL\_I2C\_Slave\_Receive()
  - HAL\_I2C\_Mem\_Write()
  - HAL\_I2C\_Mem\_Read()
  - HAL\_I2C\_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
- HAL\_I2C\_Master\_Transmit\_IT()
  - HAL\_I2C\_Master\_Receive\_IT()
  - HAL\_I2C\_Slave\_Transmit\_IT()
  - HAL\_I2C\_Slave\_Receive\_IT()
  - HAL\_I2C\_Mem\_Write\_IT()
  - HAL\_I2C\_Mem\_Read\_IT()
4. No-Blocking mode functions with DMA are :
- HAL\_I2C\_Master\_Transmit\_DMA()
  - HAL\_I2C\_Master\_Receive\_DMA()
  - HAL\_I2C\_Slave\_Transmit\_DMA()
  - HAL\_I2C\_Slave\_Receive\_DMA()
  - HAL\_I2C\_Mem\_Write\_DMA()
  - HAL\_I2C\_Mem\_Read\_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
- HAL\_I2C\_MemTxCpltCallback()
  - HAL\_I2C\_MemRxCpltCallback()
  - HAL\_I2C\_MasterTxCpltCallback()
  - HAL\_I2C\_MasterRxCpltCallback()
  - HAL\_I2C\_SlaveTxCpltCallback()
  - HAL\_I2C\_SlaveRxCpltCallback()
  - HAL\_I2C\_ErrorCallback()

This section contains the following APIs:

- [\*\*HAL\\_I2C\\_Master\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Mem\\_Write\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Mem\\_Read\(\)\*\*](#)

- [\*HAL\\_I2C\\_Mem\\_Write\\_IT\(\)\*](#)
- [\*HAL\\_I2C\\_Mem\\_Read\\_IT\(\)\*](#)
- [\*HAL\\_I2C\\_Mem\\_Write\\_DMA\(\)\*](#)
- [\*HAL\\_I2C\\_Mem\\_Read\\_DMA\(\)\*](#)
- [\*HAL\\_I2C\\_IsDeviceReady\(\)\*](#)

#### 24.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [\*HAL\\_I2C\\_GetState\(\)\*](#)
- [\*HAL\\_I2C\\_GetError\(\)\*](#)

#### 24.2.5 Detailed description of functions

##### **HAL\_I2C\_Init**

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

##### **HAL\_I2C\_DelInit**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_DelInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	Deinitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

##### **HAL\_I2C\_MspInit**

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

##### **HAL\_I2C\_MspDelInit**

Function Name	<b>void HAL_I2C_MspDelInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	I2C MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>

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
- **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
- **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	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_I2C\_Mem\_Write

Function Name	<code>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)</code>
Function Description	Write an amount of data in blocking mode to a specific memory address.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>DevAddress:</b> Target device address</li> <li><b>MemAddress:</b> Internal memory address</li> <li><b>MemAddSize:</b> Size of internal memory address</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_I2C\_Mem\_Read

Function Name	<code>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)</code>
Function Description	Read an amount of data in blocking mode from a specific memory address.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>DevAddress:</b> Target device address</li> <li><b>MemAddress:</b> Internal memory address</li> <li><b>MemAddSize:</b> Size of internal memory address</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_I2C\_IsDeviceReady

Function Name	<code>HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)</code>
---------------	---

Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>DevAddress:</b> Target device address</li> <li><b>Trials:</b> Number of trials</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function is used with Memory devices</li> </ul>

### HAL\_I2C\_Master\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit in master mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>DevAddress:</b> Target device address</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_I2C\_Master\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive in master mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>DevAddress:</b> Target device address</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_I2C\_Slave\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit in slave mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> </ul>

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

### HAL\_I2C\_Slave\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive in slave mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_I2C\_Mem\_Write\_IT

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

### HAL\_I2C\_Mem\_Read\_IT

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

**HAL\_I2C\_Master\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit in master mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>DevAddress:</b> Target device address</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_I2C\_Master\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive in master mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>DevAddress:</b> Target device address</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_I2C\_Slave\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_I2C\_Slave\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>pData:</b> Pointer to data buffer</li> </ul>

- **Size:** Amount of data to be sent
  - **HAL:** status
- Return values

### **HAL\_I2C\_Mem\_Write\_DMA**

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

### **HAL\_I2C\_Mem\_Read\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA</b> ( <b>I2C_HandleTypeDef</b> * <b>hi2c</b> , <b>uint16_t DevAddress</b> , <b>uint16_t MemAddress</b> , <b>uint16_t MemAddSize</b> , <b>uint8_t</b> * <b>pData</b> , <b>uint16_t Size</b> )
Function Description	Reads an amount of data in no-blocking mode with DMA from a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>DevAddress:</b> Target device address</li> <li>• <b>MemAddress:</b> Internal memory address</li> <li>• <b>MemAddSize:</b> Size of internal memory address</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be read</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_I2C\_EV\_IRQHandler**

Function Name	<b>void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef</b> * <b>hi2c</b> )
Function Description	This function handles I2C event interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_I2C\_ER\_IRQHandler**

Function Name	<b>void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef</b> * <b>hi2c</b> )
---------------	--

Function Description	This function handles I2C error interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_I2C\_MasterTxCpltCallback**

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

### **HAL\_I2C\_MasterRxCpltCallback**

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

### **HAL\_I2C\_SlaveTxCpltCallback**

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

### **HAL\_I2C\_SlaveRxCpltCallback**

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

### **HAL\_I2C\_MemTxCpltCallback**

Function Name	<b>void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Memory Tx Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_I2C\_MemRxCpltCallback

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

### HAL\_I2C\_ErrorCallback

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

### HAL\_I2C\_GetState

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

### HAL\_I2C\_GetError

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

## 24.3 I2C Firmware driver defines

### 24.3.1 I2C

#### *I2C addressing mode*

I2C\_ADDRESSINGMODE\_7BIT  
I2C\_ADDRESSINGMODE\_10BIT

***I2C dual addressing mode***`I2C_DUALADDRESS_DISABLE``I2C_DUALADDRESS_ENABLE`***I2C Error Code***

<code>HAL_I2C_ERROR_NONE</code>	No error
<code>HAL_I2C_ERROR_BERR</code>	BERR error
<code>HAL_I2C_ERROR_ARLO</code>	ARLO error
<code>HAL_I2C_ERROR_AF</code>	ACKF error
<code>HAL_I2C_ERROR_OVR</code>	OVR error
<code>HAL_I2C_ERROR_DMA</code>	DMA transfer error
<code>HAL_I2C_ERROR_TIMEOUT</code>	Timeout error
<code>HAL_I2C_ERROR_SIZE</code>	Size Management error

***I2C Exported Macros***`__HAL_I2C_RESET_HANDLE_STATE` **Description:**

- Reset I2C handle state.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.

**Return value:**

- None

`__HAL_I2C_ENABLE_IT` **Description:**

- Enable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable. This parameter can be one of the following values:
  - `I2C_IT_ERRI`: Errors interrupt enable
  - `I2C_IT_TCI`: Transfer complete interrupt enable
  - `I2C_IT_STOPI`: STOP detection interrupt enable
  - `I2C_IT_NACKI`: NACK received interrupt enable
  - `I2C_IT_ADDRI`: Address match interrupt enable
  - `I2C_IT_RXI`: RX interrupt enable
  - `I2C_IT_TXI`: TX interrupt enable

**Return value:**

- None

`__HAL_I2C_DISABLE_IT` **Description:**

- Disable the specified I2C interrupts.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_INTERRUPT\_\_: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C\_IT\_ERRI: Errors interrupt enable
  - I2C\_IT\_TCI: Transfer complete interrupt enable
  - I2C\_IT\_STOPI: STOP detection interrupt enable
  - I2C\_IT\_NACKI: NACK received interrupt enable
  - I2C\_IT\_ADDRI: Address match interrupt enable
  - I2C\_IT\_RXI: RX interrupt enable
  - I2C\_IT\_TXI: TX interrupt enable

**Return value:**

- None

[\\_\\_HAL\\_I2C\\_GET\\_IT\\_SOURCE](#)**Description:**

- Checks if the specified I2C interrupt source is enabled or disabled.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_INTERRUPT\_\_: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C\_IT\_ERRI: Errors interrupt enable
  - I2C\_IT\_TCI: Transfer complete interrupt enable
  - I2C\_IT\_STOPI: STOP detection interrupt enable
  - I2C\_IT\_NACKI: NACK received interrupt enable
  - I2C\_IT\_ADDRI: Address match interrupt enable
  - I2C\_IT\_RXI: RX interrupt enable
  - I2C\_IT\_TXI: TX interrupt enable

**Return value:**

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

[I2C\\_FLAG\\_MASK](#)**Description:**

- Checks whether the specified I2C flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following

values:

- I2C\_FLAG\_TXE: Transmit data register empty
- I2C\_FLAG\_TXIS: Transmit interrupt status
- I2C\_FLAG\_RXNE: Receive data register not empty
- I2C\_FLAG\_ADDR: Address matched (slave mode)
- I2C\_FLAG\_AF: Acknowledge failure received flag
- I2C\_FLAG\_STOPF: STOP detection flag
- I2C\_FLAG\_TC: Transfer complete (master mode)
- I2C\_FLAG\_TCR: Transfer complete reload
- I2C\_FLAG\_BERR: Bus error
- I2C\_FLAG\_ARLO: Arbitration lost
- I2C\_FLAG\_OVR: Overrun/Underrun
- I2C\_FLAG\_PECERR: PEC error in reception
- I2C\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
- I2C\_FLAG\_ALERT: SMBus alert
- I2C\_FLAG\_BUSY: Bus busy
- I2C\_FLAG\_DIR: Transfer direction (slave mode)

#### Return value:

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

`_HAL_I2C_GET_FLAG`

`_HAL_I2C_CLEAR_FLAG`

#### Description:

- Clears the I2C pending flags which are cleared by writing 1 in a specific bit.

#### Parameters:

- `_HANDLE_`: specifies the I2C Handle.
- `_FLAG_`: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C\_FLAG\_ADDR: Address matched (slave mode)
  - I2C\_FLAG\_AF: Acknowledge failure received flag
  - I2C\_FLAG\_STOPF: STOP detection flag
  - I2C\_FLAG\_BERR: Bus error
  - I2C\_FLAG\_ARLO: Arbitration lost
  - I2C\_FLAG\_OVR: Overrun/Underrun
  - I2C\_FLAG\_PECERR: PEC error in

- reception
- I2C\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
- I2C\_FLAG\_ALERT: SMBus alert

**Return value:**

- None

[\\_\\_HAL\\_I2C\\_ENABLE](#)

**Description:**

- Enable the specified I2C peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None

[\\_\\_HAL\\_I2C\\_DISABLE](#)

**Description:**

- Disable the specified I2C peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None

***I2C Flag definition***

[I2C\\_FLAG\\_TXE](#)

[I2C\\_FLAG\\_TXIS](#)

[I2C\\_FLAG\\_RXNE](#)

[I2C\\_FLAG\\_ADDR](#)

[I2C\\_FLAG\\_AF](#)

[I2C\\_FLAG\\_STOPF](#)

[I2C\\_FLAG\\_TC](#)

[I2C\\_FLAG\\_TCR](#)

[I2C\\_FLAG\\_BERR](#)

[I2C\\_FLAG\\_ARLO](#)

[I2C\\_FLAG\\_OVR](#)

[I2C\\_FLAG\\_PECERR](#)

[I2C\\_FLAG\\_TIMEOUT](#)

[I2C\\_FLAG\\_ALERT](#)

[I2C\\_FLAG\\_BUSY](#)

[I2C\\_FLAG\\_DIR](#)

***I2C general call addressing mode***

[I2C\\_GENERALCALL\\_DISABLE](#)

I2C\_GENERALCALL\_ENABLE

***I2C Interrupt configuration definition***

I2C\_IT\_ERRI

I2C\_IT\_TCI

I2C\_IT\_STOPI

I2C\_IT\_NACKI

I2C\_IT\_ADDRI

I2C\_IT\_RXI

I2C\_IT\_TXI

***I2C Memory Address Size***

I2C\_MEMADD\_SIZE\_8BIT

I2C\_MEMADD\_SIZE\_16BIT

***I2C nostretch mode***

I2C\_NOSTRETCH\_DISABLE

I2C\_NOSTRETCH\_ENABLE

***I2C own address2 masks***

I2C\_OA2\_NOMASK

I2C\_OA2\_MASK01

I2C\_OA2\_MASK02

I2C\_OA2\_MASK03

I2C\_OA2\_MASK04

I2C\_OA2\_MASK05

I2C\_OA2\_MASK06

I2C\_OA2\_MASK07

***I2C ReloadEndMode definition***

I2C\_RELOAD\_MODE

I2C\_AUTOEND\_MODE

I2C\_SOFTEND\_MODE

***I2C StartStopMode definition***

I2C\_NO\_STARTSTOP

I2C\_GENERATE\_STOP

I2C\_GENERATE\_START\_READ

I2C\_GENERATE\_START\_WRITE

## 25 HAL I2C Extension Driver

### 25.1 I2CEEx Firmware driver API description

#### 25.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32L0XX devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode

#### 25.1.2 How to use this driver

This driver provides functions to configure Noise Filter

1. Configure I2C Analog noise filter using the function `HAL_I2CEEx_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEEx_ConfigDigitalFilter()`
3. Configure the enable or disable of I2C Wake Up Mode using the functions : + `HAL_I2CEEx_EnableWakeUp()` + `HAL_I2CEEx_DisableWakeUp()`
4. Configure the enable or disable of fast mode plus driving capability using the functions : + `HAL_I2CEEx_EnableFastModePlus()` + `HAL_I2CEEx_DisableFastModePlus()`

#### 25.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- `HAL\_I2CEEx\_ConfigAnalogFilter\(\)`
- `HAL\_I2CEEx\_ConfigDigitalFilter\(\)`
- `HAL\_I2CEEx\_EnableWakeUp\(\)`
- `HAL\_I2CEEx\_DisableWakeUp\(\)`
- `HAL\_I2CEEx\_EnableFastModePlus\(\)`
- `HAL\_I2CEEx\_DisableFastModePlus\(\)`

#### 25.1.4 Detailed description of functions

##### `HAL_I2CEEx_ConfigAnalogFilter`

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

**HAL\_I2CEEx\_ConfigDigitalFilter**

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

**HAL\_I2CEEx\_EnableWakeUp**

Function Name	<b>HAL_StatusTypeDef HAL_I2CEEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)</b>
Function Description	Enables I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_I2CEEx\_DisableWakeUp**

Function Name	<b>HAL_StatusTypeDef HAL_I2CEEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)</b>
Function Description	Disables I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_I2CEEx\_EnableFastModePlus**

Function Name	<b>void HAL_I2CEEx_EnableFastModePlus (uint32_t ConfigFastModePlus)</b>
Function Description	Enable the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none"> <li>• <b>ConfigFastModePlus:</b> selects the pin. This parameter can be one of the I2C Fast Mode Plus values</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.</li> <li>• For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter.</li> <li>• For all I2C2 pins fast mode plus driving capability can be</li> </ul>

- enabled only by using I2C\_FASTMODEPLUS\_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be enabled only by using I2C\_FASTMODEPLUS\_I2C3 parameter.

### **HAL\_I2CEEx\_DisableFastModePlus**

Function Name	<b>void HAL_I2CEEx_DisableFastModePlus (uint32_t ConfigFastModePlus)</b>
Function Description	Disable the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none"> <li><b>ConfigFastModePlus:</b> selects the pin. This parameter can be one of the I2C Fast Mode Plus values</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.</li> <li>For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C1 parameter.</li> <li>For all I2C2 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C2 parameter.</li> <li>For all I2C3 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C3 parameter.</li> </ul>

## **25.2 I2CEEx Firmware driver defines**

### **25.2.1 I2CEEx**

#### ***I2C Analog Filter Enabling***

I2C\_ANALOGFILTER\_ENABLE

I2C\_ANALOGFILTER\_DISABLE

#### ***I2C Fast Mode Plus***

I2C\_FASTMODEPLUS\_PB6 Enable Fast Mode Plus on PB6

I2C\_FASTMODEPLUS\_PB7 Enable Fast Mode Plus on PB7

I2C\_FASTMODEPLUS\_PB8 Enable Fast Mode Plus on PB8

I2C\_FASTMODEPLUS\_PB9 Enable Fast Mode Plus on PB9

I2C\_FASTMODEPLUS\_I2C1 Enable Fast Mode Plus on I2C1 pins

I2C\_FASTMODEPLUS\_I2C2 Enable Fast Mode Plus on I2C2 pins

I2C\_FASTMODEPLUS\_I2C3 Enable Fast Mode Plus on I2C3 pins

## 26 HAL I2S Generic Driver

### 26.1 I2S Firmware driver registers structures

#### 26.1.1 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](#)

#### 26.1.2 I2S\_HandleTypeDef

##### Data Fields

- *SPI\_TypeDef \* Instance*
- *I2S\_InitTypeDef Init*
- *uint16\_t \* pTxBuffPtr*
- *\_\_IO uint16\_t TxXferSize*
- *\_\_IO uint16\_t TxXferCount*
- *uint16\_t \* pRxBuffPtr*
- *\_\_IO uint16\_t RxXferSize*
- *\_\_IO uint16\_t RxXferCount*
- *DMA\_HandleTypeDef \* hdmatx*

- **DMA\_HandleTypeDef \* hdmarx**
- **\_\_IO HAL\_LockTypeDef Lock**
- **\_\_IO HAL\_I2S\_StateTypeDef State**
- **\_\_IO uint32\_t ErrorCode**

### Field Documentation

- **SPI\_HandleTypeDef\* I2S\_HandleTypeDef::Instance**
- **I2S\_InitTypeDef I2S\_HandleTypeDef::Init**
- **uint16\_t\* I2S\_HandleTypeDef::pTxBuffPtr**
- **\_\_IO uint16\_t I2S\_HandleTypeDef::TxXferSize**
- **\_\_IO uint16\_t I2S\_HandleTypeDef::TxXferCount**
- **uint16\_t\* I2S\_HandleTypeDef::pRxBuffPtr**
- **\_\_IO uint16\_t I2S\_HandleTypeDef::RxXferSize**
- **\_\_IO uint16\_t I2S\_HandleTypeDef::RxXferCount**
- **DMA\_HandleTypeDef\* I2S\_HandleTypeDef::hdmatx**
- **DMA\_HandleTypeDef\* I2S\_HandleTypeDef::hdmarx**
- **\_\_IO HAL\_LockTypeDef I2S\_HandleTypeDef::Lock**
- **\_\_IO HAL\_I2S\_StateTypeDef I2S\_HandleTypeDef::State**
- **\_\_IO uint32\_t I2S\_HandleTypeDef::ErrorCode**

## 26.2 I2S Firmware driver API description

### 26.2.1 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. Make sure that either: External clock source is configured after setting correctly the define constant HSE\_VALUE in the stm32l0xx\_hal\_conf.h file.

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

### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_I2S\_Transmit()
- Receive an amount of data in blocking mode using HAL\_I2S\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_I2S\_Transmit\_IT()
- At transmission end of half transfer HAL\_I2S\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxHalfCpltCallback
- At transmission end of transfer HAL\_I2S\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_I2S\_Receive\_IT()
- At reception end of half transfer HAL\_I2S\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxHalfCpltCallback
- At reception end of transfer HAL\_I2S\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxCpltCallback
- In case of transfer Error, HAL\_I2S\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2S\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_I2S\_Transmit\_DMA()
- At transmission end of half transfer HAL\_I2S\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxHalfCpltCallback
- At transmission end of transfer HAL\_I2S\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_I2S\_Receive\_DMA()
- At reception end of half transfer HAL\_I2S\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxHalfCpltCallback
- At reception end of transfer HAL\_I2S\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxCpltCallback
- In case of transfer Error, HAL\_I2S\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2S\_ErrorCallback
- Pause the DMA Transfer using HAL\_I2S\_DMAPause()
- Resume the DMA Transfer using HAL\_I2S\_DMAResume()
- Stop the DMA Transfer using HAL\_I2S\_DMAStop()

### I2S HAL driver macros list

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

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



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

## 26.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze the I2Sx peripheral in simplex mode:

- User must Implement `HAL_I2S_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function `HAL_I2S_Init()` to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function `HAL_I2S_DelInit()` to restore the default configuration of the selected I2Sx periperal.

This section contains the following APIs:

- `HAL\_I2S\_Init\(\)`
- `HAL\_I2S\_DelInit\(\)`
- `HAL\_I2S\_MspInit\(\)`
- `HAL\_I2S\_MspDelInit\(\)`

## 26.2.3 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()`

#### 26.2.4 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()`

#### 26.2.5 Detailed description of functions

##### `HAL_I2S_Init`

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

##### `HAL_I2S_DeInit`

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

**HAL\_I2S\_MspInit**

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

**HAL\_I2S\_MspDelInit**

Function Name	<b>void HAL_I2S_MspDelInit (I2S_HandleTypeDef * hi2s)</b>
Function Description	I2S MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_I2S\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData:</b> a 16-bit pointer to data buffer.</li> <li>• <b>Size:</b> number of data sample to be sent:</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> <li>• This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz</li> </ul>

**HAL\_I2S\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData:</b> a 16-bit pointer to data buffer.</li> <li>• <b>Size:</b> number of data sample to be sent:</li> </ul>

- **Timeout:** Timeout duration
- **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.
  - This function can use an Audio Frequency up to 44KHz when I2S Clock Source is 32MHz

### **HAL\_I2S\_Transmit\_IT**

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_I2S_Transmit_IT<br/>(I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>   |
| Function Description | Transmit an amount of data in non-blocking mode with Interrupt.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData:</b> a 16-bit pointer to data buffer.</li> <li>• <b>Size:</b> number of data sample to be sent:</li> </ul>   |
| Return values        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>   |
| Notes                | <ul style="list-style-type: none"> <li>• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the <b>Size</b> parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the <b>Size</b> parameter means the number of 16-bit data length.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> <li>• This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz</li> </ul> |

### **HAL\_I2S\_Receive\_IT**

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_I2S_Receive_IT<br/>(I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>  |
| Function Description | Receive an amount of data in non-blocking mode with Interrupt.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData:</b> a 16-bit pointer to the Receive data buffer.</li> <li>• <b>Size:</b> number of data sample to be sent:</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>   |

---

Notes	<ul style="list-style-type: none"> <li>When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> <li>It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.</li> <li>This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz</li> </ul>
-------	---

### HAL\_I2S\_IRQHandler

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

### HAL\_I2S\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pData:</b> a 16-bit pointer to the Transmit data buffer.</li> <li><b>Size:</b> number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

### HAL\_I2S\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>

- |               |   |
|---------------|---|
|               | <ul style="list-style-type: none"> <li>• <b>pData:</b> a 16-bit pointer to the Receive data buffer.</li> <li>• <b>Size:</b> number of data sample to be sent:</li> </ul>  |
| Return values | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>  |
| Notes         | <ul style="list-style-type: none"> <li>• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul> |

### **HAL\_I2S\_DMAPause**

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

### **HAL\_I2S\_DMAResume**

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

### **HAL\_I2S\_DMAStop**

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

### **HAL\_I2S\_TxHalfCpltCallback**

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

---

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

### **HAL\_I2S\_TxCpltCallback**

Function Name	<b>void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_I2S\_RxHalfCpltCallback**

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

### **HAL\_I2S\_RxCpltCallback**

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

### **HAL\_I2S\_ErrorCallback**

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

### **HAL\_I2S\_GetState**

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

**HAL\_I2S\_GetError**

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

## 26.3 I2S Firmware driver defines

### 26.3.1 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_UDR	I2S Underrun error
HAL_I2S_ERROR_OVR	I2S Overrun error
HAL_I2S_ERROR_FRE	I2S Frame format error
HAL_I2S_ERROR_DMA	DMA transfer 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\_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 Flag definition***`I2S_FLAG_TXE``I2S_FLAG_RXNE``I2S_FLAG_UDR``I2S_FLAG_OVR``I2S_FLAG_FRE``I2S_FLAG_CHSIDE``I2S_FLAG_BSY`***I2S Interrupt configuration definition***`I2S_IT_TXE``I2S_IT_RXNE``I2S_IT_ERR`***I2S Legacy***`I2S_STANDARD_PHILLIPS`***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

## 27 HAL IRDA Generic Driver

### 27.1 IRDA Firmware driver registers structures

#### 27.1.1 IRDA\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t Parity*
- *uint16\_t Mode*
- *uint8\_t Prescaler*
- *uint16\_t PowerMode*

##### Field Documentation

- ***uint32\_t IRDA\_InitTypeDef::BaudRate***  
This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- ***uint32\_t IRDA\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [\*IRDAEx\\_Word\\_Length\*](#)
- ***uint32\_t IRDA\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of [\*IRDA\\_Parity\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint16\_t IRDA\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*IRDA\\_Transfer\\_Mode\*](#)
- ***uint8\_t IRDA\_InitTypeDef::Prescaler***  
Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.  
**Note:**Prescaler value 0 is forbidden
- ***uint16\_t IRDA\_InitTypeDef::PowerMode***  
Specifies the IRDA power mode. This parameter can be a value of [\*IRDA\\_Low\\_Power\*](#)

#### 27.1.2 IRDA\_HandleTypeDef

##### Data Fields

- *USART\_TypeDef \* Instance*
- *IRDA\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*

- *uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*
- *uint16\_t RxXferSize*
- *uint16\_t RxXferCount*
- *uint16\_t Mask*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_IRDA\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*

#### Field Documentation

- *USART\_TypeDef\* IRDA\_HandleTypeDef::Instance*
- *IRDA\_InitTypeDef IRDA\_HandleTypeDef::Init*
- *uint8\_t\* IRDA\_HandleTypeDef::pTxBuffPtr*
- *uint16\_t IRDA\_HandleTypeDef::TxXferSize*
- *uint16\_t IRDA\_HandleTypeDef::TxXferCount*
- *uint8\_t\* IRDA\_HandleTypeDef::pRxBuffPtr*
- *uint16\_t IRDA\_HandleTypeDef::RxXferSize*
- *uint16\_t IRDA\_HandleTypeDef::RxXferCount*
- *uint16\_t IRDA\_HandleTypeDef::Mask*
- *DMA\_HandleTypeDef\* IRDA\_HandleTypeDef::hdmatx*
- *DMA\_HandleTypeDef\* IRDA\_HandleTypeDef::hdmarx*
- *HAL\_LockTypeDef IRDA\_HandleTypeDef::Lock*
- *\_\_IO HAL\_IRDA\_StateTypeDef IRDA\_HandleTypeDef::State*
- *\_\_IO uint32\_t IRDA\_HandleTypeDef::ErrorCode*

## 27.2 IRDA Firmware driver API description

### 27.2.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - 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.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL\_IRDA\_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

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

## 27.2.2 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA asynchronous 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.
  - No-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_IRDA\_TxCpltCallback(), HAL\_IRDA\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL\_IRDA\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's are :
  - HAL\_IRDA\_Transmit()
  - HAL\_IRDA\_Receive()
3. Non-Blocking mode API's with Interrupt are :
  - HAL\_IRDA\_Transmit\_IT()
  - HAL\_IRDA\_Receive\_IT()
  - HAL\_IRDA\_IRQHandler()
  - IRDA\_Transmit\_IT()
  - IRDA\_Receive\_IT()
4. Non-Blocking mode functions with DMA are :
  - HAL\_IRDA\_Transmit\_DMA()
  - HAL\_IRDA\_Receive\_DMA()
5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_IRDA\_TxCpltCallback()
  - HAL\_IRDA\_RxCpltCallback()
  - HAL\_IRDA\_ErrorCallback()

This section contains the following APIs:

- [\*\*HAL\\_IRDA\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_DMAPause\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_DMAResume\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_DMAStop\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_IRQHandler\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_TxHalfCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_TxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_RxHalfCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_RxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_ErrorCallback\(\)\*\*](#)

## 27.2.3 Peripheral Control functions

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

- HAL\_IRDA\_GetState() API can be helpful to check in run-time the state of the IRDA peripheral.

- IRDA\_SetConfig() API is used to configure the IRDA communications parameters.

This section contains the following APIs:

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

## 27.2.4 Detailed description of functions

### **HAL\_IRDA\_Init**

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

### **HAL\_IRDA\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	Deinitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_IRDA\_MspInit**

Function Name	<b>void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_IRDA\_MspDeInit**

Function Name	<b>void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_IRDA\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> <li>• <b>pData:</b> pointer to data buffer</li> </ul>

- **Size:** amount of data to be sent
  - **Timeout:** Duration of the timeout
- Return values
- **HAL:** status

### **HAL\_IRDA\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> <li>• <b>Timeout:</b> Duration of the timeout</li> </ul>
Return values	• <b>HAL:</b> status

### **HAL\_IRDA\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	• <b>HAL:</b> status

### **HAL\_IRDA\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> </ul>
Return values	• <b>HAL:</b> status

### **HAL\_IRDA\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	• <b>HAL:</b> status

**HAL\_IRDA\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the IRDA parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> </ul>

**HAL\_IRDA\_DMAPause**

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

**HAL\_IRDA\_DMAResume**

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

**HAL\_IRDA\_DMAStop**

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

**HAL\_IRDA\_IRQHandler**

Function Name	<b>void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)</b>
Function Description	This function handles IRDA interrupt request.

---

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

**HAL\_IRDA\_TxCpltCallback**

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

**HAL\_IRDA\_RxCpltCallback**

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

**HAL\_IRDA\_TxHalfCpltCallback**

Function Name	<b>void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> irda handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_IRDA\_RxHalfCpltCallback**

Function Name	<b>void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Rx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> irda handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_IRDA\_ErrorCallback**

Function Name	<b>void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA error callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> IRDA handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_IRDA\_GetState**

Function Name	<b>HAL_IRDA_StateTypeDef HAL_IRDA_GetState</b>
---------------	--

**(IRDA\_HandleTypeDef \* hirda)**

Function Description	return the IRDA state
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> irda handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

**HAL\_IRDA\_GetError**

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

## 27.3 IRDA Firmware driver defines

### 27.3.1 IRDA

***IRDA DMA RX***

IRDA\_DMA\_RX\_DISABLE

IRDA\_DMA\_RX\_ENABLE

IS\_IRDA\_DMA\_RX

***IRDA DMA TX***

IRDA\_DMA\_TX\_DISABLE

IRDA\_DMA\_TX\_ENABLE

IS\_IRDA\_DMA\_TX

***IRDA Exported Constants***

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 state.

**Parameters:**

- \_HANDLE\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

`_HAL_IRDA_FLUSH_DRREGISTER`

**Description:**

- Flushes the IRDA DR register.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

`_HAL_IRDA_CLEAR_FLAG`

**Description:**

- Clears the specified IRDA pending flag.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle.
- `_FLAG_`: specifies the flag to check. This parameter can be any combination of the following values:
  - `IRDA_CLEAR_PEF`
  - `IRDA_CLEAR_FEF`
  - `IRDA_CLEAR_NEF`
  - `IRDA_CLEAR_OREF`
  - `IRDA_CLEAR_TCF`
  - `IRDA_CLEAR_IDLEF`

**Return value:**

- None

`_HAL_IRDA_CLEAR_PEFLAG`

**Description:**

- Clear the IRDA PE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle.

**Return value:**

- None

`_HAL_IRDA_CLEAR_FEFLAG`

**Description:**

- Clear the IRDA FE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle.

**Return value:**

- None

`_HAL_IRDA_CLEAR_NEFLAG`

**Description:**

- Clear the IRDA NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle.

**Return value:**

- None

`__HAL_IRDA_CLEAR_OREFLAG`

**Description:**

- Clear the IRDA ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle.

**Return value:**

- None

`__HAL_IRDA_CLEAR_IDLEFLAG`

**Description:**

- Clear the IRDA IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle.

**Return value:**

- None

`__HAL_IRDA_GET_FLAG`

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2. UART peripheral
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `IRDA_FLAG_RXACK`: Receive enable acknowledge flag
  - `IRDA_FLAG_TEACK`: Transmit enable acknowledge flag
  - `IRDA_FLAG_BUSY`: Busy flag
  - `IRDA_FLAG_ABRF`: Auto Baud rate detection flag
  - `IRDA_FLAG_ABRE`: Auto Baud rate detection error flag
  - `IRDA_FLAG_TXE`: Transmit data register empty flag
  - `IRDA_FLAG_TC`: Transmission Complete flag
  - `IRDA_FLAG_RXNE`: Receive data

- register not empty flag
- IRDA\_FLAG\_IDLE: Idle Line detection flag
- IRDA\_FLAG\_ORE: OverRun Error flag
- IRDA\_FLAG\_NE: Noise Error flag
- IRDA\_FLAG\_FE: Framing Error flag
- IRDA\_FLAG\_PE: Parity Error flag

**Return value:**

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

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

- Enable the specified IRDA interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2. UART peripheral
- \_\_INTERRUPT\_\_: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_PE: Parity Error interrupt
  - IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

[\\_\\_HAL\\_IRDA\\_DISABLE\\_IT](#)**Description:**

- Disable the specified IRDA interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_INTERRUPT\_\_: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data

- register not empty interrupt
- IRDA\_IT\_IDLE: Idle line detection interrupt
- IRDA\_IT\_PE: Parity Error interrupt
- IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

**\_HAL\_IRDA\_GET\_IT**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_IT\_\_: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_ORE: OverRun Error interrupt
  - IRDA\_IT\_NE: Noise Error interrupt
  - IRDA\_IT\_FE: Framing Error interrupt
  - IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

**\_HAL\_IRDA\_GET\_IT\_SOURCE**

- Check whether the specified IRDA interrupt source is enabled.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_IT\_\_: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt

- IRDA\_IT\_RXNE: Receive Data register not empty interrupt
- IRDA\_IT\_IDLE: Idle line detection interrupt
- IRDA\_IT\_ORE: OverRun Error interrupt
- IRDA\_IT\_NE: Noise Error interrupt
- IRDA\_IT\_FE: Framing Error interrupt
- IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

[\\_\\_HAL\\_IRDA\\_CLEAR\\_IT](#)**Description:**

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_IT\_CLEAR\_\_: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA\_CLEAR\_PEF: Parity Error Clear Flag
  - IRDA\_CLEAR\_FEF: Framing Error Clear Flag
  - IRDA\_CLEAR\_NEF: Noise detected Clear Flag
  - IRDA\_CLEAR\_OREF: OverRun Error Clear Flag
  - IRDA\_CLEAR\_TCF: Transmission Complete Clear Flag

**Return value:**

- None

[\\_\\_HAL\\_IRDA\\_SEND\\_REQ](#)**Description:**

- Set a specific IRDA request flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_REQ\_\_: specifies the request flag to set This parameter can be one of the following values:
  - IRDA\_AUTOBAUD\_REQUEST: Auto-Baud Rate Request
  - IRDA\_RXDATA\_FLUSH\_REQUEST:

- Receive Data flush Request
- IRDA\_TXDATA\_FLUSH\_REQUEST:  
Transmit data flush Request

**Return value:**

- None

\_\_HAL\_IRDA\_ONE\_BIT\_SAMPLE\_ENA  
BLE

**Description:**

- Enables the IRDA one bit sample method.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle.

**Return value:**

- None

\_\_HAL\_IRDA\_ONE\_BIT\_SAMPLE\_DISA  
BLE

**Description:**

- Disables the IRDA one bit sample method.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle.

**Return value:**

- None

\_\_HAL\_IRDA\_ENABLE

**Description:**

- Enable UART/USART associated to IRDA Handle.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

\_\_HAL\_IRDA\_DISABLE

**Description:**

- Disable UART/USART associated to IRDA Handle.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

IS\_IRDA\_BAUDRATE

**Description:**

- Ensure that IRDA Baud rate is less or

equal to maximum value.

**Parameters:**

- `__BAUDRATE__`: specifies the IRDA Baudrate set by the user.

**Return value:**

- True: or False

`IS_IRDA_PRESCALER`

**Description:**

- Ensure that IRDA prescaler value is strictly larger than 0.

**Parameters:**

- `__PRESCALER__`: specifies the IRDA prescaler value set by the user.

**Return value:**

- True: or False

***IRDA Flags***

<code>IRDA_FLAG_RXACK</code>	Receive Enable Acknowledge Flag
<code>IRDA_FLAG_TEACK</code>	Transmit Enable Acknowledge Flag
<code>IRDA_FLAG_BUSY</code>	Busy Flag
<code>IRDA_FLAG_ABRF</code>	Auto-Baud Rate Flag
<code>IRDA_FLAG_ABRE</code>	Auto-Baud Rate Error
<code>IRDA_FLAG_TXE</code>	Transmit Data Register Empty
<code>IRDA_FLAG_TC</code>	Transmission Complete
<code>IRDA_FLAG_RXNE</code>	Read Data Register Not Empty
<code>IRDA_FLAG_ORE</code>	OverRun Error
<code>IRDA_FLAG_NE</code>	Noise detected Flag
<code>IRDA_FLAG_FE</code>	Framing Error
<code>IRDA_FLAG_PE</code>	Parity Error

***IRDA Interruption mask***

`IRDA_IT_MASK`

***IRDA Interrupt definition***

<code>IRDA_IT_PE</code>
<code>IRDA_IT_TXE</code>
<code>IRDA_IT_TC</code>
<code>IRDA_IT_RXNE</code>
<code>IRDA_IT_IDLE</code>
<code>IRDA_IT_ERR</code>
<code>IRDA_IT_ORE</code>

IRDA\_IT\_NE

IRDA\_IT\_FE

***IRDA Interrupt clear flag***

IRDA\_CLEAR\_PEF      Parity Error Clear Flag

IRDA\_CLEAR\_FEF      Framing Error Clear Flag

IRDA\_CLEAR\_NEF      Noise detected Clear Flag

IRDA\_CLEAR\_OREF      OverRun Error Clear Flag

IRDA\_CLEAR\_TCF      Transmission Complete Clear Flag

IRDA\_CLEAR\_IDLEF      IDLE line detected Clear Flag

***IRDA low power***

IRDA\_POWERMODE\_NORMAL

IRDA\_POWERMODE\_LOWPOWER

IS\_IRDA\_POWERMODE

***IRDA Mode***

IRDA\_MODE\_DISABLE

IRDA\_MODE\_ENABLE

IS\_IRDA\_MODE

***IRDA One bit***

IRDA\_ONE\_BIT\_SAMPLE\_DISABLE

IRDA\_ONE\_BIT\_SAMPLE\_ENABLE

IS\_IRDA\_ONE\_BIT\_SAMPLE

***IRDA Parity***

IRDA\_PARITY\_NONE

IRDA\_PARITY\_EVEN

IRDA\_PARITY\_ODD

IS\_IRDA\_PARITY

***IRDA Request parameters***

IRDA\_AUTOBAUD\_REQUEST      Auto-Baud Rate Request

IRDA\_RXDATA\_FLUSH\_REQUEST      Receive Data flush Request

IRDA\_TXDATA\_FLUSH\_REQUEST      Transmit data flush Request

IS\_IRDA\_REQUEST\_PARAMETER

***IRDA State***

IRDA\_STATE\_DISABLE

IRDA\_STATE\_ENABLE

IS\_IRDA\_STATE

***IRDA transfer mode***

IRDA\_MODE\_RX  
IRDA\_MODE\_TX  
IRDA\_MODE\_TX\_RX  
IS\_IRDA\_TX\_RX\_MODE

## 28 HAL IRDA Extension Driver

### 28.1 IRDAEx Firmware driver defines

#### 28.1.1 IRDAEx

##### *IRDAEx Exported Macros*

IRDA\_GETCLOCKSOURCE

##### Description:

- Reports the IRDA clock source.

##### Parameters:

- \_\_HANDLE\_\_: specifies the UART Handle
- \_\_CLOCKSOURCE\_\_: output variable

##### Return value:

- IRDA: clocking source, written in \_\_CLOCKSOURCE\_\_.

IRDA\_MASK\_COMPUTATION

##### Description:

- Reports the mask to apply to retrieve the received data according to the word length and to the parity bits activation.

##### Parameters:

- \_\_HANDLE\_\_: specifies the IRDA Handle

##### Return value:

- mask: to apply to USART RDR register value.

##### *IRDAEx Word length*

IRDA\_WORDLENGTH\_7B

IRDA\_WORDLENGTH\_8B

IRDA\_WORDLENGTH\_9B

IS\_IRDA\_WORD\_LENGTH

## 29 HAL IWDG Generic Driver

### 29.1 IWDG Firmware driver registers structures

#### 29.1.1 IWDG\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*
- *uint32\_t Window*

##### Field Documentation

- ***uint32\_t IWDG\_InitTypeDef::Prescaler***  
Select the prescaler of the IWDG. This parameter can be a value of [\*IWDG\\_Prescaler\*](#)
- ***uint32\_t IWDG\_InitTypeDef::Reload***  
Specifies the IWDG down-counter reload value. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF
- ***uint32\_t IWDG\_InitTypeDef::Window***  
Specifies the window value to be compared to the down-counter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF

#### 29.1.2 IWDG\_HandleTypeDef

##### Data Fields

- *IWDG\_TypeDef \* Instance*
- *IWDG\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_IWDG\_StateTypeDef State*

##### Field Documentation

- ***IWDG\_TypeDef\* IWDG\_HandleTypeDef::Instance***  
Register base address
- ***IWDG\_InitTypeDef IWDG\_HandleTypeDef::Init***  
IWDG required parameters
- ***HAL\_LockTypeDef IWDG\_HandleTypeDef::Lock***  
IWDG Locking object
- ***\_\_IO HAL\_IWDG\_StateTypeDef IWDG\_HandleTypeDef::State***  
IWDG communication state

## 29.2 IWDG Firmware driver API description

### 29.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by its own dedicated Low-Speed clock (LSI) and thus stays active even if the main clock fails. Once the IWDG is started, the LSI is forced ON and cannot be disabled (LSI cannot be disabled too), and the counter starts counting down from the reset value of 0xFFFF. When it reaches the end of count value (0x000) a system reset is generated.
- The IWDG counter should be refreshed at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC\_CSR register can be used to inform when an IWDG reset occurs.

Min-max timeout value @32KHz (LSI): ~0.512ms / ~32.0s The IWDG timeout may vary due to LSI frequency dispersion. STM32L0xx 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.

### 29.2.2 How to use this driver

If Window option is disabled

- Use IWDG using HAL\_IWDG\_Init() function to :
  - Enable write access to IWDG\_PR, IWDG\_RLR.
  - Configure the IWDG prescaler, counter reload value. This reload value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
- Use IWDG using HAL\_IWDG\_Start() function to :
  - Reload IWDG counter with value defined in the IWDG\_RLR register.
  - Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.

if Window option is enabled:

- Use IWDG using HAL\_IWDG\_Start() function to enable IWDG downcounter
- Use IWDG using HAL\_IWDG\_Init() function to :
  - Enable write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.
  - Configure the IWDG prescaler, reload value and window value.
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.

### IWDG HAL driver macros list

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

- \_\_HAL\_IWDG\_START: Enable the IWDG peripheral
- \_\_HAL\_IWDG\_RELOAD\_COUNTER: Reloads IWDG counter with value defined in the reload register

- `IWDG_ENABLE_WRITE_ACCESS` : Enable write access to IWDG\_PR and IWDG\_RLR registers
- `_HAL_IWDG_DISABLE_WRITE_ACCESS` : Disable write access to IWDG\_PR and IWDG\_RLR registers
- `_HAL_IWDG_GET_FLAG`: Get the selected IWDG's flag status

### 29.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the `IWDG_InitTypeDef` and create the associated handle
- Manage Window option
- Initialize the IWDG MSP

This section contains the following APIs:

- `HAL\_IWDG\_Init\(\)`
- `HAL\_IWDG\_MspInit\(\)`

### 29.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.

This section contains the following APIs:

- `HAL\_IWDG\_Start\(\)`
- `HAL\_IWDG\_Refresh\(\)`

### 29.2.5 Peripheral State functions

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

This section contains the following APIs:

- `HAL\_IWDG\_GetState\(\)`

### 29.2.6 Detailed description of functions

#### HAL\_IWDG\_Init

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

#### HAL\_IWDG\_MspInit

Function Name	<code>void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)</code>
Function Description	Initializes the IWDG MSP.

Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg:</b> pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_IWDG\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Starts the IWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg:</b> : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_IWDG\_Refresh**

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Refreshes the IWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg:</b> : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_IWDG\_GetState**

Function Name	<b>HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Returns the IWDG state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg:</b> : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

**29.3 IWDG Firmware driver defines****29.3.1 IWDG*****IWDG Disable***

IWDG\_WINDOW\_DISABLE

***IWDG Exported Constants***

IS\_IWDG\_KR

IS\_IWDG\_PRESCALER

IS\_IWDG\_RELOAD

IS\_IWDG\_WINDOW

**IWDG Exported Macros**

`_HAL_IWDG_RESET_HANDLE_STATE`

**Description:**

- Reset IWDG handle state.

**Parameters:**

- `_HANDLE_`: IWDG handle

**Return value:**

- None

`_HAL_IWDG_START`

**Description:**

- Enables the IWDG peripheral.

**Parameters:**

- `_HANDLE_`: IWDG handle

**Return value:**

- None

`_HAL_IWDG_RELOAD_COUNTER`

**Description:**

- Reloads IWDG counter with value defined in the reload register (write access to IWDG\_PR and IWDG\_RLR registers disabled).

**Parameters:**

- `_HANDLE_`: IWDG handle

**Return value:**

- None

`IWDG_ENABLE_WRITE_ACCESS`

**Description:**

- Enables write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.

**Parameters:**

- `_HANDLE_`: IWDG handle

**Return value:**

- None

`IWDG_DISABLE_WRITE_ACCESS`

**Description:**

- Disables write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.

**Parameters:**

- `_HANDLE_`: IWDG handle

**Return value:**

- None

[\\_\\_HAL\\_IWDG\\_GET\\_FLAG](#)**Description:**

- Gets the selected IWDG's flag status.

**Parameters:**

- \_\_HANDLE\_\_: IWDG handle
- \_\_FLAG\_\_: specifies the flag to check.  
This parameter can be one of the following values:
  - IWDG\_FLAG\_PVU: Watchdog counter reload value update flag
  - IWDG\_FLAG\_RVU: Watchdog counter prescaler value flag
  - IWDG\_FLAG\_WVU: Watchdog counter window value flag

**Return value:**

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

***IWDG Flag definition***

IWDG_FLAG_PVU	Watchdog counter prescaler value update flag
IWDG_FLAG_RVU	Watchdog counter reload value update flag
IWDG_FLAG_WVU	Watchdog counter window value update Flag

***IWDG Prescaler***

IWDG_PRESCALER_4	IWDG prescaler set to 4
IWDG_PRESCALER_8	IWDG prescaler set to 8
IWDG_PRESCALER_16	IWDG prescaler set to 16
IWDG_PRESCALER_32	IWDG prescaler set to 32
IWDG_PRESCALER_64	IWDG prescaler set to 64
IWDG_PRESCALER_128	IWDG prescaler set to 128
IWDG_PRESCALER_256	IWDG prescaler set to 256

***IWDG key***

IWDG_KEY_RELOAD	IWDG Reload Counter Enable
IWDG_KEY_ENABLE	IWDG Peripheral Enable
IWDG_KEY_WRITE_ACCESS_ENABLE	IWDG KR Write Access Enable
IWDG_KEY_WRITE_ACCESS_DISABLE	IWDG KR Write Access Disable

## 30 HAL LCD Generic Driver

### 30.1 LCD Firmware driver registers structures

#### 30.1.1 LCD\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Divider*
- *uint32\_t Duty*
- *uint32\_t Bias*
- *uint32\_t VoltageSource*
- *uint32\_t Contrast*
- *uint32\_t DeadTime*
- *uint32\_t PulseOnDuration*
- *uint32\_t HighDrive*
- *uint32\_t BlinkMode*
- *uint32\_t BlinkFrequency*
- *uint32\_t MuxSegment*

##### Field Documentation

- ***uint32\_t LCD\_InitTypeDef::Prescaler***  
Configures the LCD Prescaler. This parameter can be one value of [\*\*LCD\\_Prescaler\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::Divider***  
Configures the LCD Divider. This parameter can be one value of [\*\*LCD\\_Divider\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::Duty***  
Configures the LCD Duty. This parameter can be one value of [\*\*LCD\\_Duty\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::Bias***  
Configures the LCD Bias. This parameter can be one value of [\*\*LCD\\_Bias\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::VoltageSource***  
Selects the LCD Voltage source. This parameter can be one value of [\*\*LCD\\_Voltage\\_Source\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::Contrast***  
Configures the LCD Contrast. This parameter can be one value of [\*\*LCD\\_Contrast\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::DeadTime***  
Configures the LCD Dead Time. This parameter can be one value of [\*\*LCD\\_DeadTime\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::PulseOnDuration***  
Configures the LCD Pulse On Duration. This parameter can be one value of [\*\*LCD\\_PulseOnDuration\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::HighDrive***  
Configures the LCD High Drive. This parameter can be one value of [\*\*LCD\\_HighDrive\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::BlinkMode***  
Configures the LCD Blink Mode. This parameter can be one value of [\*\*LCD\\_BlinkMode\*\*](#)
- ***uint32\_t LCD\_InitTypeDef::BlinkFrequency***  
Configures the LCD Blink frequency. This parameter can be one value of [\*\*LCD\\_BlinkFrequency\*\*](#)

- ***uint32\_t LCD\_InitTypeDef::MuxSegment***  
Enable or disable mux segment. This parameter can be one value of ***LCD\_MuxSegment***

### 30.1.2 LCD\_HandleTypeDef

#### Data Fields

- ***LCD\_TypeDef \* Instance***
- ***LCD\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_LCD\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***LCD\_TypeDef\* LCD\_HandleTypeDef::Instance***
- ***LCD\_InitTypeDef LCD\_HandleTypeDef::Init***
- ***HAL\_LockTypeDef LCD\_HandleTypeDef::Lock***
- ***\_\_IO HAL\_LCD\_StateTypeDef LCD\_HandleTypeDef::State***
- ***\_\_IO uint32\_t LCD\_HandleTypeDef::ErrorCode***

## 30.2 LCD Firmware driver API description

### 30.2.1 How to use this driver

The LCD HAL driver can be used as follow:

1. Declare a LCD\_HandleTypeDef handle structure.
2. Prepare the initialization of the LCD low level resources by implementing your HAL\_LCD\_MspInit() API:
  - a. Enable the LCDCLK (same as RTCCLK): to configure the RTCCLK/LCDCLK, use the RCC function HAL\_RCCEx\_PeriphCLKConfig, indicating here RCC\_PERIPHCLK\_LCD and the selected clock source (HSE, LSI or LSE)
  - b. The frequency generator allows you to achieve various LCD frame rates starting from an LCD input clock frequency (LCDCLK) which can vary from 32 kHz up to 1 MHz.
  - c. LCD pins configuration: - Enable the clock for the LCD GPIOs - Configure these LCD pins as alternate function no-pull.
  - d. Enable the LCD interface clock.
3. Set the Prescaler, Divider, Blink mode, Blink Frequency Duty, Bias, Voltage Source, Dead Time, Pulse On Duration and Contrast in the hlcd Init structure.
4. Initialize the LCD registers by calling the HAL\_LCD\_Init() API.
  - a. The HAL\_LCD\_Init() API configures the low level Hardware (GPIO, CLOCK, ...etc) by calling the user customized HAL\_LCD\_MspInit() API.
5. After calling the HAL\_LCD\_Init() the LCD RAM memory is cleared
6. Optionally you can update the LCD configuration using these macros:
  - a. LCD High Drive using the \_\_HAL\_LCD\_HIGHDIVER\_ENABLE() and \_\_HAL\_LCD\_HIGHDIVER\_DISABLE() macros
  - b. LCD Pulse ON Duration using the \_\_HAL\_LCD\_PULSEONDURATION\_CONFIG() macro

- c. LCD Dead Time using the \_\_HAL\_LCD\_DEADTIME\_CONFIG() macro
- d. The LCD Blink mode and frequency using the \_\_HAL\_LCD\_BLINK\_CONFIG() macro
- e. The LCD Contrast using the \_\_HAL\_LCD\_CONTRAST\_CONFIG() macro
- 7. Write to the LCD RAM memory using the HAL\_LCD\_Write() API, this API can be called several times to update the different LCD RAM registers before calling HAL\_LCD\_UpdateDisplayRequest() API.
- 8. The HAL\_LCD\_Clear() API can be used to clear the LCD RAM memory.
- 9. When the LCD RAM memory is updated, enable the update display request calling the HAL\_LCD\_UpdateDisplayRequest() API.

LCD and low power modes: The LCD remain active during STOP mode.

### 30.2.2 Initialization and Configuration functions

This section contains the following APIs:

- [`HAL\_LCD\_DelInit\(\)`](#)
- [`HAL\_LCD\_Init\(\)`](#)
- [`HAL\_LCD\_MspDelInit\(\)`](#)
- [`HAL\_LCD\_MspInit\(\)`](#)

### 30.2.3 IO operation functions

Using its double buffer memory the LCD controller ensures the coherency of the displayed information without having to use interrupts to control LCD\_RAM modification. The application software can access the first buffer level (LCD\_RAM) through the APB interface. Once it has modified the LCD\_RAM using the HAL\_LCD\_Write() API, it sets the UDR flag in the LCD\_SR register using the HAL\_LCD\_UpdateDisplayRequest() API. This UDR flag (update display request) requests the updated information to be moved into the second buffer level (LCD\_DISPLAY). This operation is done synchronously with the frame (at the beginning of the next frame), until the update is completed, the LCD\_RAM is write protected and the UDR flag stays high. Once the update is completed another flag (UDD - Update Display Done) is set and generates an interrupt if the UDDIE bit in the LCD\_FCR register is set. The time it takes to update LCD\_DISPLAY is, in the worst case, one odd and one even frame. The update will not occur (UDR = 1 and UDD = 0) until the display is enabled (LCDEN = 1).

This section contains the following APIs:

- [`HAL\_LCD\_Write\(\)`](#)
- [`HAL\_LCD\_Clear\(\)`](#)
- [`HAL\_LCD\_UpdateDisplayRequest\(\)`](#)

### 30.2.4 Peripheral State functions

This subsection provides a set of functions allowing to control the LCD:

- HAL\_LCD\_GetState() API can be helpful to check in run-time the state of the LCD peripheral State.
- HAL\_LCD\_GetError() API to return the LCD error code.

This section contains the following APIs:

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

### 30.2.5 Detailed description of functions

#### HAL\_LCD\_Delnit

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

#### HAL\_LCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_LCD_Init (LCD_HandleTypeDef * hlcd)</b>
Function Description	Initializes the LCD peripheral according to the specified parameters in the LCD_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function can be used only when the LCD is disabled. The LCD HighDrive can be enabled/disabled using related macros up to user.</li> </ul>

#### HAL\_LCD\_MsplInit

Function Name	<b>void HAL_LCD_MsplInit (LCD_HandleTypeDef * hlcd)</b>
Function Description	LCD MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### HAL\_LCD\_MspDelnit

Function Name	<b>void HAL_LCD_MspDelnit (LCD_HandleTypeDef * hlcd)</b>
Function Description	LCD MSP Delnit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### HAL\_LCD\_Write

Function Name	<b>HAL_StatusTypeDef HAL_LCD_Write (LCD_HandleTypeDef * hlcd, uint32_t RAMRegisterIndex, uint32_t RAMRegisterMask, uint32_t Data)</b>
Function Description	Writes a word in the specific LCD RAM.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> <li>• <b>RAMRegisterIndex:</b> specifies the LCD RAM Register. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LCD_RAM_REGISTER0: LCD RAM Register 0</li> </ul> </li> </ul>

- LCD\_RAM\_REGISTER1: LCD RAM Register 1
  - LCD\_RAM\_REGISTER2: LCD RAM Register 2
  - LCD\_RAM\_REGISTER3: LCD RAM Register 3
  - LCD\_RAM\_REGISTER4: LCD RAM Register 4
  - LCD\_RAM\_REGISTER5: LCD RAM Register 5
  - LCD\_RAM\_REGISTER6: LCD RAM Register 6
  - LCD\_RAM\_REGISTER7: LCD RAM Register 7
  - LCD\_RAM\_REGISTER8: LCD RAM Register 8
  - LCD\_RAM\_REGISTER9: LCD RAM Register 9
  - LCD\_RAM\_REGISTER10: LCD RAM Register 10
  - LCD\_RAM\_REGISTER11: LCD RAM Register 11
  - LCD\_RAM\_REGISTER12: LCD RAM Register 12
  - LCD\_RAM\_REGISTER13: LCD RAM Register 13
  - LCD\_RAM\_REGISTER14: LCD RAM Register 14
  - LCD\_RAM\_REGISTER15: LCD RAM Register 15
  - **RAMRegisterMask:** specifies the LCD RAM Register Data Mask.
  - **Data:** specifies LCD Data Value to be written.
  - **None:**
- Return values**
- Notes**
- For LCD glass COM\*SEG as 8\*40 for example, the LCD common terminals COM[0,7] are mapped on 32bits LCD\_RAM\_REGISTER[0,14] according to rules: COM(n) spread on LCD\_RAM\_REGISTER(2\*n) and LCD\_RAM\_REGISTER(2\*n+1).The segment terminals SEG[0,39] of COM(n) correspond to LSB bits of related LCD\_RAM\_REGISTER(2\*n)[0,31] and LCD\_RAM\_REGISTER(2\*n+1)[0,7]

### HAL\_LCD\_Clear

Function Name	<b>HAL_StatusTypeDef HAL_LCD_Clear (LCD_HandleTypeDef * hlcd)</b>
Function Description	Clears the LCD RAM registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_LCD\_UpdateDisplayRequest

Function Name	<b>HAL_StatusTypeDef HAL_LCD_UpdateDisplayRequest (LCD_HandleTypeDef * hlcd)</b>
Function Description	Enables the Update Display Request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Each time software modifies the LCD_RAM it must set the UDR bit to transfer the updated data to the second level buffer. The UDR bit stays set until the end of the update and during this time the LCD_RAM is write protected.</li> <li>• When the display is disabled, the update is performed for all LCD_DISPLAY locations. When the display is enabled, the</li> </ul>

update is performed only for locations for which commons are active (depending on DUTY). For example if DUTY = 1/2, only the LCD\_DISPLAY of COM0 and COM1 will be updated.

### **HAL\_LCD\_GetState**

Function Name	<b>HAL_LCD_StateTypeDef HAL_LCD_GetState (LCD_HandleTypeDef * hlcd)</b>
Function Description	Returns the LCD state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

### **HAL\_LCD\_GetError**

Function Name	<b>uint32_t HAL_LCD_GetError (LCD_HandleTypeDef * hlcd)</b>
Function Description	Return the LCD error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>LCD:</b> Error Code</li> </ul>

### **LCD\_WaitForSynchro**

Function Name	<b>HAL_StatusTypeDef LCD_WaitForSynchro (LCD_HandleTypeDef * hlcd)</b>
Function Description	Waits until the LCD FCR register is synchronized in the LCDCLK domain.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlcd:</b> LCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## **30.3 LCD Firmware driver defines**

### **30.3.1 LCD**

#### ***LCD Bias***

LCD_BIAS_1_4	1/4 Bias
LCD_BIAS_1_2	1/2 Bias
LCD_BIAS_1_3	1/3 Bias
<b>IS_LCD_BIAS</b>	

#### ***LCD Blink Frequency***

LCD_BLINKFREQUENCY_DIV8	The Blink frequency = fLCD/8
LCD_BLINKFREQUENCY_DIV16	The Blink frequency = fLCD/16
LCD_BLINKFREQUENCY_DIV32	The Blink frequency = fLCD/32
LCD_BLINKFREQUENCY_DIV64	The Blink frequency = fLCD/64
LCD_BLINKFREQUENCY_DIV128	The Blink frequency = fLCD/128

LCD_BLINKFREQUENCY_DIV256	The Blink frequency = fLCD/256
LCD_BLINKFREQUENCY_DIV512	The Blink frequency = fLCD/512
LCD_BLINKFREQUENCY_DIV1024	The Blink frequency = fLCD/1024
<b>IS_LCD_BLINK_FREQUENCY</b>	
<b>LCD Blink Mode</b>	
LCD_BLINKMODE_OFF	Blink disabled
LCD_BLINKMODE_SEG0_COM0	Blink enabled on SEG[0], COM[0] (1 pixel)
LCD_BLINKMODE_SEG0_ALLCOM	Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
LCD_BLINKMODE_ALLSEG_ALLCOM	Blink enabled on all SEG and all COM (all pixels)
<b>IS_LCD_BLINK_MODE</b>	
<b>LCD Voltage output buffer enable</b>	
LCD_VOLTBUFOUT_DISABLE	Voltage output buffer disabled
LCD_VOLTBUFOUT_ENABLE	BUFEN[1] Voltage output buffer enabled
<b>IS_LCD_VOLTBUFOUT</b>	
<b>LCD Contrast</b>	
LCD_CONTRASTLEVEL_0	Maximum Voltage = 2.60V
LCD_CONTRASTLEVEL_1	Maximum Voltage = 2.73V
LCD_CONTRASTLEVEL_2	Maximum Voltage = 2.86V
LCD_CONTRASTLEVEL_3	Maximum Voltage = 2.99V
LCD_CONTRASTLEVEL_4	Maximum Voltage = 3.12V
LCD_CONTRASTLEVEL_5	Maximum Voltage = 3.25V
LCD_CONTRASTLEVEL_6	Maximum Voltage = 3.38V
LCD_CONTRASTLEVEL_7	Maximum Voltage = 3.51V
<b>IS_LCD_CONTRAST</b>	
<b>LCD Dead Time</b>	
LCD_DEADTIME_0	No dead Time
LCD_DEADTIME_1	One Phase between different couple of Frame
LCD_DEADTIME_2	Two Phase between different couple of Frame
LCD_DEADTIME_3	Three Phase between different couple of Frame
LCD_DEADTIME_4	Four Phase between different couple of Frame
LCD_DEADTIME_5	Five Phase between different couple of Frame
LCD_DEADTIME_6	Six Phase between different couple of Frame
LCD_DEADTIME_7	Seven Phase between different couple of Frame
<b>IS_LCD_DEAD_TIME</b>	
<b>LCD Divider</b>	
LCD_DIVIDER_16	LCD frequency = CLKPS/16

LCD_DIVIDER_17	LCD frequency = CLKPS/17
LCD_DIVIDER_18	LCD frequency = CLKPS/18
LCD_DIVIDER_19	LCD frequency = CLKPS/19
LCD_DIVIDER_20	LCD frequency = CLKPS/20
LCD_DIVIDER_21	LCD frequency = CLKPS/21
LCD_DIVIDER_22	LCD frequency = CLKPS/22
LCD_DIVIDER_23	LCD frequency = CLKPS/23
LCD_DIVIDER_24	LCD frequency = CLKPS/24
LCD_DIVIDER_25	LCD frequency = CLKPS/25
LCD_DIVIDER_26	LCD frequency = CLKPS/26
LCD_DIVIDER_27	LCD frequency = CLKPS/27
LCD_DIVIDER_28	LCD frequency = CLKPS/28
LCD_DIVIDER_29	LCD frequency = CLKPS/29
LCD_DIVIDER_30	LCD frequency = CLKPS/30
LCD_DIVIDER_31	LCD frequency = CLKPS/31

IS\_LCD\_DIVIDER

**LCD Duty**

LCD_DUTY_STATIC	Static duty
LCD_DUTY_1_2	1/2 duty
LCD_DUTY_1_3	1/3 duty
LCD_DUTY_1_4	1/4 duty
LCD_DUTY_1_8	1/8 duty

IS\_LCD\_DUTY

**LCD Error Code**

HAL_LCD_ERROR_NONE	No error
HAL_LCD_ERROR_FCRSF	Synchro flag timeout error
HAL_LCD_ERROR_UDR	Update display request flag timeout error
HAL_LCD_ERROR_UDD	Update display done flag timeout error
HAL_LCD_ERROR_ENS	LCD enabled status flag timeout error
HAL_LCD_ERROR_RDY	LCD Booster ready timeout error

**LCD Exported Macros**\_\_HAL\_LCD\_RESET\_HANDLE\_STATE    **Description:**

- Reset LCD handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the LCD Handle.

**Return value:**

- None

`__HAL_LCD_ENABLE`**Description:**

- macros to enables or disables the LCD

**Parameters:**

- `__HANDLE__`: specifies the LCD Handle.

**Return value:**

- None

`__HAL_LCD_DISABLE`**Description:**

- macros to enables or disables the Voltage output buffer

**Parameters:**

- `__HANDLE__`: specifies the LCD Handle.

**Return value:**

- None

`__HAL_LCD_VOLTOUTBUFFER_DISABLE`**Description:**

- Macros to enable or disable the low resistance divider.

**Parameters:**

- `__HANDLE__`: specifies the LCD Handle.

**Return value:**

- None

**Notes:**

- When this mode is enabled, the PulseOn Duration (PON) have to be programmed to 1/CK\_PS (LCD\_PULSEONDURATION\_1).

`__HAL_LCD_HIGHDIVER_DISABLE`**Description:**

- Macro to configure the LCD pulses on duration.

**Parameters:**

- `__HANDLE__`: specifies the LCD Handle.
- `__DURATION__`: specifies the LCD pulse on duration in terms of CK\_PS (prescaled LCD clock period) pulses. This parameter can be one of the following values:
  - `LCD_PULSEONDURATION_0`: 0 pulse
  - `LCD_PULSEONDURATION_1`: Pulse

- ON duration = 1/CK\_PS
- LCD\_PULSEONDURATION\_2: Pulse ON duration = 2/CK\_PS
- LCD\_PULSEONDURATION\_3: Pulse ON duration = 3/CK\_PS
- LCD\_PULSEONDURATION\_4: Pulse ON duration = 4/CK\_PS
- LCD\_PULSEONDURATION\_5: Pulse ON duration = 5/CK\_PS
- LCD\_PULSEONDURATION\_6: Pulse ON duration = 6/CK\_PS
- LCD\_PULSEONDURATION\_7: Pulse ON duration = 7/CK\_PS

**Return value:**

- None

**\_HAL\_LCD\_DEADTIME\_CONFIG**

- Macro to configure the LCD dead time.

**Parameters:**

- \_HANDLE\_: specifies the LCD Handle.
- \_DEADTIME\_: specifies the LCD dead time. This parameter can be one of the following values:
  - LCD\_DEADTIME\_0: No dead Time
  - LCD\_DEADTIME\_1: One Phase between different couple of Frame
  - LCD\_DEADTIME\_2: Two Phase between different couple of Frame
  - LCD\_DEADTIME\_3: Three Phase between different couple of Frame
  - LCD\_DEADTIME\_4: Four Phase between different couple of Frame
  - LCD\_DEADTIME\_5: Five Phase between different couple of Frame
  - LCD\_DEADTIME\_6: Six Phase between different couple of Frame
  - LCD\_DEADTIME\_7: Seven Phase between different couple of Frame

**Return value:**

- None

**\_HAL\_LCD\_CONTRAST\_CONFIG**

- Macro to configure the LCD Contrast.

**Parameters:**

- \_HANDLE\_: specifies the LCD Handle.
- \_CONTRAST\_: specifies the LCD Contrast. This parameter can be one of the following values:
  - LCD\_CONTRASTLEVEL\_0: Maximum Voltage = 2.60V

- LCD\_CONTRASTLEVEL\_1:  
Maximum Voltage = 2.73V
- LCD\_CONTRASTLEVEL\_2:  
Maximum Voltage = 2.86V
- LCD\_CONTRASTLEVEL\_3:  
Maximum Voltage = 2.99V
- LCD\_CONTRASTLEVEL\_4:  
Maximum Voltage = 3.12V
- LCD\_CONTRASTLEVEL\_5:  
Maximum Voltage = 3.25V
- LCD\_CONTRASTLEVEL\_6:  
Maximum Voltage = 3.38V
- LCD\_CONTRASTLEVEL\_7:  
Maximum Voltage = 3.51V

**Return value:**

- None

**\_HAL\_LCD\_BLINK\_CONFIG****Description:**

- Macro to configure the LCD Blink mode and Blink frequency.

**Parameters:**

- \_HANDLE\_: specifies the LCD Handle.
- \_BLINKMODE\_: specifies the LCD blink mode. This parameter can be one of the following values:
  - LCD\_BLINKMODE\_OFF: Blink disabled
  - LCD\_BLINKMODE\_SEG0\_COM0: Blink enabled on SEG[0], COM[0] (1 pixel)
  - LCD\_BLINKMODE\_SEG0\_ALLCOM: Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
  - LCD\_BLINKMODE\_ALLSEG\_ALLCOM: Blink enabled on all SEG and all COM (all pixels)
- \_BLINKFREQUENCY\_: specifies the LCD blink frequency.
  - LCD\_BLINKFREQUENCY\_DIV8: The Blink frequency = fLcd/8
  - LCD\_BLINKFREQUENCY\_DIV16: The Blink frequency = fLcd/16
  - LCD\_BLINKFREQUENCY\_DIV32: The Blink frequency = fLcd/32
  - LCD\_BLINKFREQUENCY\_DIV64: The Blink frequency = fLcd/64
  - LCD\_BLINKFREQUENCY\_DIV128: The Blink frequency = fLcd/128
  - LCD\_BLINKFREQUENCY\_DIV256: The Blink frequency = fLcd/256
  - LCD\_BLINKFREQUENCY\_DIV512:

- The Blink frequency =  $f_{LCD}/512$
- LCD\_BLINKFREQUENCY\_DIV1024:
- The Blink frequency =  $f_{LCD}/1024$

**Return value:**

- None

**\_HAL\_LCD\_ENABLE\_IT****Description:**

- Enables or disables the specified LCD interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the LCD Handle.
- \_\_INTERRUPT\_\_: specifies the LCD interrupt source to be enabled or disabled. This parameter can be one of the following values:
  - LCD\_IT\_SOF: Start of Frame Interrupt
  - LCD\_IT\_UDD: Update Display Done Interrupt

**Return value:**

- None

**\_HAL\_LCD\_DISABLE\_IT****\_HAL\_LCD\_GET\_IT\_SOURCE****Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the LCD Handle.
- \_\_IT\_\_: specifies the LCD interrupt source to check. This parameter can be one of the following values:
  - LCD\_IT\_SOF: Start of Frame Interrupt
  - LCD\_IT\_UDD: Update Display Done Interrupt.

**Return value:**

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

**Notes:**

- If the device is in STOP mode (PCLK not provided) UDD will not generate an interrupt even if UDDIE = 1. If the display is not enabled the UDD interrupt will never occur.

**\_HAL\_LCD\_GET\_FLAG****Description:**

- Checks whether the specified LCD flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the LCD Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `LCD_FLAG_ENS`: LCD Enabled flag. It indicates the LCD controller status.

**Return value:**

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

**Notes:**

- The ENS bit is set immediately when the LCDEN bit in the LCD\_CR goes from 0 to 1. On deactivation it reflects the real status of LCD so it becomes 0 at the end of the last displayed frame. `LCD_FLAG_SOF`: Start of Frame flag. This flag is set by hardware at the beginning of a new frame, at the same time as the display data is updated. `LCD_FLAG_UDR`: Update Display Request flag. `LCD_FLAG_UDD`: Update Display Done flag. `LCD_FLAG_RDY`: Step\_up converter Ready flag. It indicates the status of the step-up converter. `LCD_FLAG_FCRSF`: LCD Frame Control Register Synchronization Flag. This flag is set by hardware each time the LCD\_FCR register is updated in the LCDCLK domain.

[`\_\_HAL\_LCD\_CLEAR\_FLAG`](#)**Description:**

- Clears the specified LCD pending flag.

**Parameters:**

- `__HANDLE__`: specifies the LCD Handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `LCD_FLAG_SOF`: Start of Frame Interrupt
  - `LCD_FLAG_UDD`: Update Display Done Interrupt

**Return value:**

- None

***LCD Flag***[`LCD\_FLAG\_ENS`](#)[`LCD\_FLAG\_SOF`](#)[`LCD\_FLAG\_UDR`](#)

LCD\_FLAG\_UDD

LCD\_FLAG\_RDY

LCD\_FLAG\_FCRSF

**LCD HighDrive**

LCD\_HIGHDRIVE\_0 Low resistance Drive

LCD\_HIGHDRIVE\_1 High resistance Drive

IS\_LCD\_HIGHDRIVE

**LCD Interrupts**

LCD\_IT\_SOF

LCD\_IT\_UDD

**LCD Mux Segment**

LCD\_MUXSEGMENT\_DISABLE SEG pin multiplexing disabled

LCD\_MUXSEGMENT\_ENABLE SEG[31:28] are multiplexed with SEG[43:40]

IS\_LCD\_MUXSEGMENT

**LCD Prescaler**

LCD\_PRESCALER\_1 CLKPS = LCDCLK

LCD\_PRESCALER\_2 CLKPS = LCDCLK/2

LCD\_PRESCALER\_4 CLKPS = LCDCLK/4

LCD\_PRESCALER\_8 CLKPS = LCDCLK/8

LCD\_PRESCALER\_16 CLKPS = LCDCLK/16

LCD\_PRESCALER\_32 CLKPS = LCDCLK/32

LCD\_PRESCALER\_64 CLKPS = LCDCLK/64

LCD\_PRESCALER\_128 CLKPS = LCDCLK/128

LCD\_PRESCALER\_256 CLKPS = LCDCLK/256

LCD\_PRESCALER\_512 CLKPS = LCDCLK/512

LCD\_PRESCALER\_1024 CLKPS = LCDCLK/1024

LCD\_PRESCALER\_2048 CLKPS = LCDCLK/2048

LCD\_PRESCALER\_4096 CLKPS = LCDCLK/4096

LCD\_PRESCALER\_8192 CLKPS = LCDCLK/8192

LCD\_PRESCALER\_16384 CLKPS = LCDCLK/16384

LCD\_PRESCALER\_32768 CLKPS = LCDCLK/32768

IS\_LCD\_PRESCALER

**LCD Pulse On Duration**

LCD\_PULSEONDURATION\_0 Pulse ON duration = 0 pulse

LCD\_PULSEONDURATION\_1 Pulse ON duration = 1/CK\_PS

LCD\_PULSEONDURATION\_2 Pulse ON duration = 2/CK\_PS

LCD\_PULSEONDURATION\_3      Pulse ON duration = 3/CK\_PS  
LCD\_PULSEONDURATION\_4      Pulse ON duration = 4/CK\_PS  
LCD\_PULSEONDURATION\_5      Pulse ON duration = 5/CK\_PS  
LCD\_PULSEONDURATION\_6      Pulse ON duration = 6/CK\_PS  
LCD\_PULSEONDURATION\_7      Pulse ON duration = 7/CK\_PS  
  
IS\_LCD\_PULSE\_ON\_DURATION

***LCD RAMRegister***

LCD\_RAM\_REGISTER0      LCD RAM Register 0  
LCD\_RAM\_REGISTER1      LCD RAM Register 1  
LCD\_RAM\_REGISTER2      LCD RAM Register 2  
LCD\_RAM\_REGISTER3      LCD RAM Register 3  
LCD\_RAM\_REGISTER4      LCD RAM Register 4  
LCD\_RAM\_REGISTER5      LCD RAM Register 5  
LCD\_RAM\_REGISTER6      LCD RAM Register 6  
LCD\_RAM\_REGISTER7      LCD RAM Register 7  
LCD\_RAM\_REGISTER8      LCD RAM Register 8  
LCD\_RAM\_REGISTER9      LCD RAM Register 9  
LCD\_RAM\_REGISTER10      LCD RAM Register 10  
LCD\_RAM\_REGISTER11      LCD RAM Register 11  
LCD\_RAM\_REGISTER12      LCD RAM Register 12  
LCD\_RAM\_REGISTER13      LCD RAM Register 13  
LCD\_RAM\_REGISTER14      LCD RAM Register 14  
LCD\_RAM\_REGISTER15      LCD RAM Register 15  
  
IS\_LCD\_RAM\_REGISTER

***LCD Voltage Source***

LCD\_VOLTAGESOURCE\_INTERNAL      Internal voltage source for the LCD  
LCD\_VOLTAGESOURCE\_EXTERNAL      External voltage source for the LCD  
  
IS\_LCD\_VOLTAGE\_SOURCE

## 31 HAL LPTIM Generic Driver

### 31.1 LPTIM Firmware driver registers structures

#### 31.1.1 LPTIM\_ClockConfigTypeDef

##### Data Fields

- *uint32\_t Source*
- *uint32\_t Prescaler*

##### Field Documentation

- ***uint32\_t LPTIM\_ClockConfigTypeDef::Source***  
Selects the clock source. This parameter can be a value of [\*LPTIM\\_Clock\\_Source\*](#)
- ***uint32\_t LPTIM\_ClockConfigTypeDef::Prescaler***  
Specifies the counter clock Prescaler. This parameter can be a value of [\*LPTIM\\_Clock\\_Prescaler\*](#)

#### 31.1.2 LPTIM\_ULPClockConfigTypeDef

##### Data Fields

- *uint32\_t Polarity*
- *uint32\_t SampleTime*

##### Field Documentation

- ***uint32\_t LPTIM\_ULPClockConfigTypeDef::Polarity***  
Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of [\*LPTIM\\_Clock\\_Polarity\*](#)
- ***uint32\_t LPTIM\_ULPClockConfigTypeDef::SampleTime***  
Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of [\*LPTIM\\_Clock\\_Sample\\_Time\*](#)

#### 31.1.3 LPTIM\_TriggerConfigTypeDef

##### Data Fields

- *uint32\_t Source*
- *uint32\_t ActiveEdge*
- *uint32\_t SampleTime*

### Field Documentation

- ***uint32\_t LPTIM\_TriggerConfigTypeDef::Source***  
Selects the Trigger source. This parameter can be a value of [\*\*LPTIM\\_Trigger\\_Source\*\*](#)
- ***uint32\_t LPTIM\_TriggerConfigTypeDef::ActiveEdge***  
Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [\*\*LPTIM\\_External\\_Trigger\\_Polarity\*\*](#)
- ***uint32\_t LPTIM\_TriggerConfigTypeDef::SampleTime***  
Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [\*\*LPTIM\\_Trigger\\_Sample\\_Time\*\*](#)

## 31.1.4 LPTIM\_InitTypeDef

### Data Fields

- ***LPTIM\_ClockConfigTypeDef Clock***
- ***LPTIM\_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM\_TriggerConfigTypeDef Trigger***
- ***uint32\_t OutputPolarity***
- ***uint32\_t UpdateMode***
- ***uint32\_t CounterSource***

### Field Documentation

- ***LPTIM\_ClockConfigTypeDef LPTIM\_InitTypeDef::Clock***  
Specifies the clock parameters
- ***LPTIM\_ULPClockConfigTypeDef LPTIM\_InitTypeDef::UltraLowPowerClock***  
Specifies the Ultra Low Power clock parameters
- ***LPTIM\_TriggerConfigTypeDef LPTIM\_InitTypeDef::Trigger***  
Specifies the Trigger parameters
- ***uint32\_t LPTIM\_InitTypeDef::OutputPolarity***  
Specifies the Output polarity. This parameter can be a value of [\*\*LPTIM\\_Output\\_Polarity\*\*](#)
- ***uint32\_t LPTIM\_InitTypeDef::UpdateMode***  
Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of [\*\*LPTIM Updating Mode\*\*](#)
- ***uint32\_t LPTIM\_InitTypeDef::CounterSource***  
Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of [\*\*LPTIM Counter\\_Source\*\*](#)

## 31.1.5 LPTIM\_HandleTypeDef

### Data Fields

- ***LPTIM\_TypeDef \* Instance***
- ***LPTIM\_InitTypeDef Init***

- ***HAL\_StatusTypeDef Status***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_LPTIM\_StateTypeDef State***

#### Field Documentation

- ***LPTIM\_TypeDef\* LPTIM\_HandleTypeDef::Instance***  
Register base address
- ***LPTIM\_InitTypeDef LPTIM\_HandleTypeDef::Init***  
LPTIM required parameters
- ***HAL\_StatusTypeDef LPTIM\_HandleTypeDef::Status***  
LPTIM peripheral status
- ***HAL\_LockTypeDef LPTIM\_HandleTypeDef::Lock***  
LPTIM locking object
- ***\_\_IO HAL\_LPTIM\_StateTypeDef LPTIM\_HandleTypeDef::State***  
LPTIM peripheral state

## 31.2 LPTIM Firmware driver API description

### 31.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM\_InitTypeDef and creates the associated handle.
- Deinitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- Deinitialize LPTIM MSP.

This section contains the following APIs:

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

### 31.2.2 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- `HAL_LPTIM_PWM_Start()`
- `HAL_LPTIM_PWM_Stop()`
- `HAL_LPTIM_PWM_Start_IT()`
- `HAL_LPTIM_PWM_Stop_IT()`
- `HAL_LPTIM_OnePulse_Start()`
- `HAL_LPTIM_OnePulse_Stop()`
- `HAL_LPTIM_OnePulse_Start_IT()`
- `HAL_LPTIM_OnePulse_Stop_IT()`
- `HAL_LPTIM_SetOnce_Start()`
- `HAL_LPTIM_SetOnce_Stop()`
- `HAL_LPTIM_SetOnce_Start_IT()`
- `HAL_LPTIM_SetOnce_Stop_IT()`
- `HAL_LPTIM_Encoder_Start()`
- `HAL_LPTIM_Encoder_Stop()`
- `HAL_LPTIM_Encoder_Start_IT()`
- `HAL_LPTIM_Encoder_Stop_IT()`
- `HAL_LPTIM_TimeOut_Start()`
- `HAL_LPTIM_TimeOut_Stop()`
- `HAL_LPTIM_TimeOut_Start_IT()`
- `HAL_LPTIM_TimeOut_Stop_IT()`
- `HAL_LPTIM_Counter_Start()`
- `HAL_LPTIM_Counter_Stop()`
- `HAL_LPTIM_Counter_Start_IT()`
- `HAL_LPTIM_Counter_Stop_IT()`

### 31.2.3 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare) value.

This section contains the following APIs:

- `HAL_LPTIM_ReadCounter()`
- `HAL_LPTIM_ReadAutoReload()`
- `HAL_LPTIM_ReadCompare()`

### 31.2.4 LPTIM IRQ handler

This section provides LPTIM IRQ handler function.

This section contains the following APIs:

- `HAL_LPTIM_IRQHandler()`
- `HAL_LPTIM_CompareMatchCallback()`
- `HAL_LPTIM_AutoReloadMatchCallback()`
- `HAL_LPTIM_TriggerCallback()`
- `HAL_LPTIM_CompareWriteCallback()`
- `HAL_LPTIM_AutoReloadWriteCallback()`
- `HAL_LPTIM_DirectionUpCallback()`
- `HAL_LPTIM_DirectionDownCallback()`

### 31.2.5 Peripheral State functions

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

This section contains the following APIs:

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

### 31.2.6 Detailed description of functions

#### **HAL\_LPTIM\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Initializes the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### **HAL\_LPTIM\_DelInit**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_DelInit (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Deinitializes the LPTIM peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### **HAL\_LPTIM\_MspInit**

Function Name	<b>void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Initializes the LPTIM MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### **HAL\_LPTIM\_MspDelInit**

Function Name	<b>void HAL_LPTIM_MspDelInit (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Deinitializes LPTIM MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### **HAL\_LPTIM\_PWM\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)</b>
Function Description	Starts the LPTIM PWM generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Pulse:</b> : Specifies the compare value. This parameter must</li> </ul>

be a value between 0x0000 and 0xFFFF.

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

### HAL\_LPTIM\_PWM\_Stop

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_PWM\_Stop  
(LPTIM\_HandleTypeDef \* htim)**

Function Description      Stops the LPTIM PWM generation.

Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : LPTIM handle</li> </ul>
------------	---

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

### HAL\_LPTIM\_PWM\_Start\_IT

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_PWM\_Start\_IT  
(LPTIM\_HandleTypeDef \* htim, uint32\_t Period, uint32\_t Pulse)**

Function Description      Starts the LPTIM PWM generation in interrupt mode.

Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : LPTIM handle</li> <li><b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF</li> <li><b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF</li> </ul>
------------	---

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

### HAL\_LPTIM\_PWM\_Stop\_IT

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_PWM\_Stop\_IT  
(LPTIM\_HandleTypeDef \* htim)**

Function Description      Stops the LPTIM PWM generation in interrupt mode.

Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : LPTIM handle</li> </ul>
------------	---

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

### HAL\_LPTIM\_OnePulse\_Start

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_OnePulse\_Start  
(LPTIM\_HandleTypeDef \* htim, uint32\_t Period, uint32\_t Pulse)**

Function Description      Starts the LPTIM One pulse generation.

Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : LPTIM handle</li> <li><b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li><b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
------------	---

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

**HAL\_LPTIM\_OnePulse\_Stop**

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_OnePulse\_Stop  
(LPTIM\_HandleTypeDef \* hltim)**

Function Description      Stops the LPTIM One pulse generation.

Parameters      • **hltim:** : LPTIM handle

Return values      • **HAL:** status

**HAL\_LPTIM\_OnePulse\_Start\_IT**

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_OnePulse\_Start\_IT  
(LPTIM\_HandleTypeDef \* hltim, uint32\_t Period, uint32\_t Pulse)**

Function Description      Starts the LPTIM One pulse generation in interrupt mode.

Parameters      • **hltim:** : LPTIM handle  
• **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.  
• **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values      • **HAL:** status

**HAL\_LPTIM\_OnePulse\_Stop\_IT**

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_OnePulse\_Stop\_IT  
(LPTIM\_HandleTypeDef \* hltim)**

Function Description      Stops the LPTIM One pulse generation in interrupt mode.

Parameters      • **hltim:** : LPTIM handle

Return values      • **HAL:** status

**HAL\_LPTIM\_SetOnce\_Start**

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_SetOnce\_Start  
(LPTIM\_HandleTypeDef \* hltim, uint32\_t Period, uint32\_t Pulse)**

Function Description      Starts the LPTIM in Set once mode.

Parameters      • **hltim:** : LPTIM handle  
• **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.  
• **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values      • **HAL:** status

**HAL\_LPTIM\_SetOnce\_Stop**

Function Name      **HAL\_StatusTypeDef HAL\_LPTIM\_SetOnce\_Stop  
(LPTIM\_HandleTypeDef \* hltim)**

Function Description      Stops the LPTIM Set once mode.

---

Parameters	<ul style="list-style-type: none"> <li><b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_LPTIM\_SetOnce\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)</b>
Function Description	Starts the LPTIM Set once mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>hlptim:</b> : LPTIM handle</li> <li><b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li><b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_LPTIM\_SetOnce\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Stops the LPTIM Set once mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_LPTIM\_Encoder\_Start

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)</b>
Function Description	Starts the Encoder interface.
Parameters	<ul style="list-style-type: none"> <li><b>hlptim:</b> : LPTIM handle</li> <li><b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_LPTIM\_Encoder\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Stops the Encoder interface.
Parameters	<ul style="list-style-type: none"> <li><b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_LPTIM\_Encoder\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)</b>
---------------	---

Function Description	Starts the Encoder interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_LPTIM\_Encoder\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Stops the Encoder interface in nterrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>

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

### **HAL\_LPTIM\_TimeOut\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)</b>
Function Description	Starts the Timeout function.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Timeout:</b> : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_LPTIM\_TimeOut\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Stops the Timeout function.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>

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

### **HAL\_LPTIM\_TimeOut\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)</b>
Function Description	Starts the Timeout function in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Timeout:</b> : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>

0xFFFF.

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

### **HAL\_LPTIM\_TimeOut\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hltim)</b>
---------------	--

Function Description Stops the Timeout function in interrupt mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
------------	--

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

### **HAL\_LPTIM\_Counter\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period)</b>
---------------	---

Function Description Starts the Counter mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
------------	--

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

### **HAL\_LPTIM\_Counter\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hltim)</b>
---------------	---

Function Description Stops the Counter mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
------------	--

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

### **HAL\_LPTIM\_Counter\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period)</b>
---------------	--

Function Description Starts the Counter mode in interrupt mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
------------	--

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

### **HAL\_LPTIM\_Counter\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hltim)</b>
---------------	--

Function Description Stops the Counter mode in interrupt mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
------------	--

Return values • **HAL:** status

### **HAL\_LPTIM\_ReadCounter**

Function Name **uint32\_t HAL\_LPTIM\_ReadCounter (LPTIM\_HandleTypeDef \* hltim)**

Function Description This function returns the current counter value.

Parameters • **hltim:** LPTIM handle

Return values • **Counter:** value.

### **HAL\_LPTIM\_ReadAutoReload**

Function Name **uint32\_t HAL\_LPTIM\_ReadAutoReload (LPTIM\_HandleTypeDef \* hltim)**

Function Description This function return the current Autoreload (Period) value.

Parameters • **hltim:** LPTIM handle

Return values • **Autoreload:** value.

### **HAL\_LPTIM\_ReadCompare**

Function Name **uint32\_t HAL\_LPTIM\_ReadCompare (LPTIM\_HandleTypeDef \* hltim)**

Function Description This function return the current Compare (Pulse) value.

Parameters • **hltim:** LPTIM handle

Return values • **Compare:** value.

### **HAL\_LPTIM\_IRQHandler**

Function Name **void HAL\_LPTIM\_IRQHandler (LPTIM\_HandleTypeDef \* hltim)**

Function Description This function handles LPTIM interrupt request.

Parameters • **hltim:** LPTIM handle

Return values • **None:**

### **HAL\_LPTIM\_CompareMatchCallback**

Function Name **void HAL\_LPTIM\_CompareMatchCallback (LPTIM\_HandleTypeDef \* hltim)**

Function Description Compare match callback in non blocking mode.

Parameters • **hltim:** : LPTIM handle

Return values • **None:**

### **HAL\_LPTIM\_AutoReloadMatchCallback**

Function Name **void HAL\_LPTIM\_AutoReloadMatchCallback**

**(LPTIM\_HandleTypeDef \* hltim)**

Function Description	Autoreload match callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_LPTIM\_TriggerCallback**

Function Name	<b>void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Trigger detected callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_LPTIM\_CompareWriteCallback**

Function Name	<b>void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Compare write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_LPTIM\_AutoReloadWriteCallback**

Function Name	<b>void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Autoreload write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_LPTIM\_DirectionUpCallback**

Function Name	<b>void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Direction counter changed from Down to Up callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_LPTIM\_DirectionDownCallback**

Function Name	<b>void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hltim)</b>
Function Description	Direction counter changed from Up to Down callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltim:</b> : LPTIM handle</li> </ul>

- 
- Return values
    - None:

### **HAL\_LPTIM\_GetState**

Function Name      **HAL\_LPTIM\_StateTypeDef HAL\_LPTIM\_GetState  
(LPTIM\_HandleTypeDef \* hltim)**

Function Description      Returns the LPTIM state.

Parameters     

- **hltim:** LPTIM handle

Return values     

- **HAL:** state

## **31.3 LPTIM Firmware driver defines**

### **31.3.1 LPTIM**

#### ***LPTIM Clock configuration structure***

**LPTIM\_EXTI\_LINE\_WAKEUPTIMER\_EVENT**      External interrupt line 29 Connected to the LPTIM EXTI Line

#### ***Clock polarity***

LPTIM\_CLOCKPOLARITY\_RISING

LPTIM\_CLOCKPOLARITY\_FALLING

LPTIM\_CLOCKPOLARITY\_RISING\_FALLING

#### ***Prescaler***

LPTIM\_PRESCALER\_DIV1

LPTIM\_PRESCALER\_DIV2

LPTIM\_PRESCALER\_DIV4

LPTIM\_PRESCALER\_DIV8

LPTIM\_PRESCALER\_DIV16

LPTIM\_PRESCALER\_DIV32

LPTIM\_PRESCALER\_DIV64

LPTIM\_PRESCALER\_DIV128

#### ***Clock sample time***

LPTIM\_CLOCKSAMPLETIME\_DIRECTTRANSITION

LPTIM\_CLOCKSAMPLETIME\_2TRANSITIONS

LPTIM\_CLOCKSAMPLETIME\_4TRANSITIONS

LPTIM\_CLOCKSAMPLETIME\_8TRANSITIONS

#### ***Clock source***

LPTIM\_CLOCKSOURCE\_APBCLOCK\_LPOS

LPTIM\_CLOCKSOURCE\_ULPTIM

#### ***Counter source***

LPTIM\_COUNTERSOURCE\_INTERNAL

LPTIM\_COUNTERSOURCE\_EXTERNAL

**LPTIM Exported constants**

IS\_LPTIM\_AUTORELOAD  
IS\_LPTIM\_COMPARE  
IS\_LPTIM\_CLOCK\_SOURCE  
IS\_LPTIM\_CLOCK\_PRESCALER  
IS\_LPTIM\_CLOCK\_PRESCALERDIV1  
IS\_LPTIM\_OUTPUT\_POLARITY  
IS\_LPTIM\_CLOCK\_SAMPLE\_TIME  
IS\_LPTIM\_CLOCK\_POLARITY  
IS\_LPTIM\_EXT\_TRG\_POLARITY  
IS\_LPTIM\_TRIG\_SAMPLE\_TIME  
IS\_LPTIM\_UPDATE\_MODE  
IS\_LPTIM\_COUNTER\_SOURCE  
IS\_LPTIM\_PERIOD  
IS\_LPTIM\_PULSE

**LPTIM Exported Macros**

`_HAL_LPTIM_RESET_HANDLE_STATE`

**Description:**

- Reset LPTIM handle state.

**Parameters:**

- `_HANDLE_`: LPTIM handle

**Return value:**

- None

`_HAL_LPTIM_ENABLE`

**Description:**

- Enable/Disable the LPTIM peripheral.

**Parameters:**

- `_HANDLE_`: LPTIM handle

**Return value:**

- None

`_HAL_LPTIM_DISABLE`

`_HAL_LPTIM_START_CONTINUOUS`

**Description:**

- Starts the LPTIM peripheral in Continuous or in single mode.

**Parameters:**

- `__HANDLE__`: DMA handle

**Return value:**

- None

`__HAL_LPTIM_START_SINGLE`  
`__HAL_LPTIM_AUTORELOAD_SET`

**Description:**

- Writes the passed parameter in the Autoreload register.

**Parameters:**

- `__HANDLE__`: LPTIM handle
- `__VALUE__`: Autoreload value

**Return value:**

- None

`__HAL_LPTIM_COMPARE_SET`

**Description:**

- Writes the passed parameter in the Compare register.

**Parameters:**

- `__HANDLE__`: LPTIM handle
- `__VALUE__`: Compare value

**Return value:**

- None

`__HAL_LPTIM_GET_FLAG`

**Description:**

- Checks whether the specified LPTIM flag is set or not.

**Parameters:**

- `__HANDLE__`: LPTIM handle
- `__FLAG__`: LPTIM flag to check  
This parameter can be a value of:
  - `LPTIM_FLAG_DOWN` : Counter direction change up Flag.
  - `LPTIM_FLAG_UP` : Counter direction change down to up Flag.
  - `LPTIM_FLAG_ARROK` : Autoreload register update OK Flag.

- LPTIM\_FLAG\_CMPOK : Compare register update OK Flag.
- LPTIM\_FLAG\_EXTTRIG : External trigger edge event Flag.
- LPTIM\_FLAG\_ARRM : Autoreload match Flag.
- LPTIM\_FLAG\_CMPM : Compare match Flag.

**Return value:**

- The state of the specified flag (SET or RESET).

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

- Clears the specified LPTIM flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): LPTIM handle.
- [\\_\\_FLAG\\_\\_](#): LPTIM flag to clear. This parameter can be a value of:
  - LPTIM\_FLAG\_DOWN : Counter direction change up Flag.
  - LPTIM\_FLAG\_UP : Counter direction change down to up Flag.
  - LPTIM\_FLAG\_ARROK : Autoreload register update OK Flag.
  - LPTIM\_FLAG\_CMPOK : Compare register update OK Flag.
  - LPTIM\_FLAG\_EXTTRIG : External trigger edge event Flag.
  - LPTIM\_FLAG\_ARRM : Autoreload match Flag.
  - LPTIM\_FLAG\_CMPM : Compare match Flag.

**Return value:**

- None.

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

- Enable the specified LPTIM interrupt.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): LPTIM handle.
- [\\_\\_INTERRUPT\\_\\_](#): LPTIM interrupt to set. This parameter can be a value of:

- LPTIM\_IT\_DOWN : Counter direction change up Interrupt.
- LPTIM\_IT\_UP : Counter direction change down to up Interrupt.
- LPTIM\_IT\_ARROK : Autoreload register update OK Interrupt.
- LPTIM\_IT\_CMPOK : Compare register update OK Interrupt.
- LPTIM\_IT\_EXTTRIG : External trigger edge event Interrupt.
- LPTIM\_IT\_ARRM : Autoreload match Interrupt.
- LPTIM\_IT\_CMPM : Compare match Interrupt.

**Return value:**

- None.

**\_HAL\_LPTIM\_DISABLE\_IT****Description:**

- Disable the specified LPTIM interrupt.

**Parameters:**

- \_HANDLE\_: : LPTIM handle.
- \_INTERRUPT\_: : LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM\_IT\_DOWN : Counter direction change up Interrupt.
  - LPTIM\_IT\_UP : Counter direction change down to up Interrupt.
  - LPTIM\_IT\_ARROK : Autoreload register update OK Interrupt.
  - LPTIM\_IT\_CMPOK : Compare register update OK Interrupt.
  - LPTIM\_IT\_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM\_IT\_ARRM : Autoreload match Interrupt.
  - LPTIM\_IT\_CMPM : Compare match Interrupt.

**Return value:**

- None.

**\_HAL\_LPTIM\_GET\_IT\_SOURCE****Description:**

- Checks whether the specified LPTIM interrupt is set or not.

**Parameters:**

- `__HANDLE__`: : LPTIM handle.
- `__INTERRUPT__`: : LPTIM interrupt to check. This parameter can be a value of:
  - `LPTIM_IT_DOWN` : Counter direction change up Interrupt.
  - `LPTIM_IT_UP` : Counter direction change down to up Interrupt.
  - `LPTIM_IT_ARROK` : Autoreload register update OK Interrupt.
  - `LPTIM_IT_CMPOK` : Compare register update OK Interrupt.
  - `LPTIM_IT_EXTTRIG` : External trigger edge event Interrupt.
  - `LPTIM_IT_ARRM` : Autoreload match Interrupt.
  - `LPTIM_IT_CMPM` : Compare match Interrupt.

**Return value:**

- Interrupt: status.

**Description:**

- Enable interrupt on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None

**Description:**

- Disable interrupt on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None

**Description:**

- Enable event on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Disable event on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Enable falling edge trigger on the

LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE`

**Description:**

- Disable falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE`

**Description:**

- Enable rising edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE`

**Description:**

- Disable rising edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE`

**Description:**

- Enable rising & falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE`

**Description:**

- Disable rising & falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_LPTIM_WAKEUPTIMER_EXTI_GET_FLAG`

**Description:**

- Check whether the LPTIM Wake-up Timer associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

<code>__HAL_LPTIM_WAKEUPTIMER_EXTI_CLEA R_FLAG</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Clear the LPTIM Wake-up Timer associated Exti line flag.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None.</li></ul>
<code>__HAL_LPTIM_WAKEUPTIMER_EXTI_GENE RATE_SWIT</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Generate a Software interrupt on the LPTIM Wake-up Timer associated Exti line.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None.</li></ul>
<b>Trigger polarity</b>	
<code>LPTIM_ACTIVEEDGE_RISING</code>	
<code>LPTIM_ACTIVEEDGE_FALLING</code>	
<code>LPTIM_ACTIVEEDGE_RISING_FALLING</code>	
<b>Flag definition</b>	
<code>LPTIM_FLAG_DOWN</code>	
<code>LPTIM_FLAG_UP</code>	
<code>LPTIM_FLAG_ARROK</code>	
<code>LPTIM_FLAG_CMPOK</code>	
<code>LPTIM_FLAG_EXTTRIG</code>	
<code>LPTIM_FLAG_ARRM</code>	
<code>LPTIM_FLAG_CMPM</code>	
<b>Interrupts definition</b>	
<code>LPTIM_IT_DOWN</code>	
<code>LPTIM_IT_UP</code>	
<code>LPTIM_IT_ARROK</code>	
<code>LPTIM_IT_CMPOK</code>	
<code>LPTIM_IT_EXTTRIG</code>	
<code>LPTIM_IT_ARRM</code>	
<code>LPTIM_IT_CMPM</code>	
<b>Output polarity</b>	
<code>LPTIM_OUTPUTPOLARITY_HIGH</code>	
<code>LPTIM_OUTPUTPOLARITY_LOW</code>	
<b>Trigger sample time</b>	
<code>LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION</code>	
<code>LPTIM_TRIGSAMPLETIME_2TRANSITIONS</code>	

LPTIM\_TRIGSAMPLETIME\_4TRANSITIONS

LPTIM\_TRIGSAMPLETIME\_8TRANSITIONS

***Trigger source***

LPTIM\_TRIGSOURCE\_SOFTWARE

LPTIM\_TRIGSOURCE\_0

LPTIM\_TRIGSOURCE\_1

LPTIM\_TRIGSOURCE\_2

LPTIM\_TRIGSOURCE\_3

LPTIM\_TRIGSOURCE\_4

LPTIM\_TRIGSOURCE\_5

LPTIM\_TRIGSOURCE\_6

LPTIM\_TRIGSOURCE\_7

***Updating mode***

LPTIM\_UPDATE\_IMMEDIATE

LPTIM\_UPDATE\_ENDOFPERIOD

## 32 HAL PCD Generic Driver

### 32.1 PCD Firmware driver registers structures

#### 32.1.1 PCD\_InitTypeDef

##### Data Fields

- *uint32\_t dev\_endpoints*
- *uint32\_t speed*
- *uint32\_t ep0\_mps*
- *uint32\_t phy\_iface*
- *uint32\_t Sof\_enable*
- *uint32\_t low\_power\_enable*
- *uint32\_t lpm\_enable*
- *uint32\_t battery\_charging\_enable*

##### Field Documentation

- ***uint32\_t PCD\_InitTypeDef::dev\_endpoints***  
Device Endpoints number. This parameter depends on the used USB core. This parameter must be a number between Min\_Data = 1 and Max\_Data = 15
- ***uint32\_t PCD\_InitTypeDef::speed***  
USB Core speed. This parameter can be any value of [PCD\\_Speed](#)
- ***uint32\_t PCD\_InitTypeDef::ep0\_mps***  
Set the Endpoint 0 Max Packet size. This parameter can be any value of [PCD\\_USB\\_EP0\\_MPS](#)
- ***uint32\_t PCD\_InitTypeDef::phy\_iface***  
Select the used PHY interface. This parameter can be any value of [PCD\\_USB\\_Core\\_PHY](#)
- ***uint32\_t PCD\_InitTypeDef::Sof\_enable***  
Enable or disable the output of the SOF signal. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t PCD\_InitTypeDef::low\_power\_enable***  
Enable or disable Low Power mode This parameter can be set to ENABLE or DISABLE
- ***uint32\_t PCD\_InitTypeDef::lpm\_enable***  
Enable or disable Link Power Management. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t PCD\_InitTypeDef::battery\_charging\_enable***  
Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

#### 32.1.2 PCD\_EPTypeDef

##### Data Fields

- *uint8\_t num*

- *uint8\_t is\_in*
- *uint8\_t is\_stall*
- *uint8\_t type*
- *uint16\_t pmaaddress*
- *uint16\_t pmaaddr0*
- *uint16\_t pmaaddr1*
- *uint8\_t doublebuffer*
- *uint32\_t maxpacket*
- *uint8\_t \*xfer\_buff*
- *uint32\_t xfer\_len*
- *uint32\_t xfer\_count*

#### Field Documentation

- ***uint8\_t PCD\_EPTypedef::num***  
Endpoint number This parameter must be a number between Min\_Data = 1 and Max\_Data = 15
- ***uint8\_t PCD\_EPTypedef::is\_in***  
Endpoint direction This parameter must be a number between Min\_Data = 0 and Max\_Data = 1
- ***uint8\_t PCD\_EPTypedef::is\_stall***  
Endpoint stall condition This parameter must be a number between Min\_Data = 0 and Max\_Data = 1
- ***uint8\_t PCD\_EPTypedef::type***  
Endpoint type This parameter can be any value of [PCD\\_USB\\_EP\\_Type](#)
- ***uint16\_t PCD\_EPTypedef::pmaaddress***  
PMA Address This parameter can be any value between Min\_addr = 0 and Max\_addr = 1K
- ***uint16\_t PCD\_EPTypedef::pmaaddr0***  
PMA Address0 This parameter can be any value between Min\_addr = 0 and Max\_addr = 1K
- ***uint16\_t PCD\_EPTypedef::pmaaddr1***  
PMA Address1 This parameter can be any value between Min\_addr = 0 and Max\_addr = 1K
- ***uint8\_t PCD\_EPTypedef::doublebuffer***  
Double buffer enable This parameter can be 0 or 1
- ***uint32\_t PCD\_EPTypedef::maxpacket***  
Endpoint Max packet size This parameter must be a number between Min\_Data = 0 and Max\_Data = 64KB
- ***uint8\_t\* PCD\_EPTypedef::xfer\_buff***  
Pointer to transfer buffer
- ***uint32\_t PCD\_EPTypedef::xfer\_len***  
Current transfer length
- ***uint32\_t PCD\_EPTypedef::xfer\_count***  
Partial transfer length in case of multi packet transfer

### 32.1.3 PCD\_HandleTypeDef

#### Data Fields

- *PCD\_TypeDef \*Instance*

- *PCD\_InitTypeDef Init*
- *\_IO uint8\_t USB\_Address*
- *PCD\_EPTTypeDef IN\_ep*
- *PCD\_EPTTypeDef OUT\_ep*
- *HAL\_LockTypeDef Lock*
- *\_IO PCD\_StateTypeDef State*
- *uint32\_t Setup*
- *void \* pData*

#### Field Documentation

- *PCD\_TypeDef\* PCD\_HandleTypeDef::Instance*  
Register base address
- *PCD\_InitTypeDef PCD\_HandleTypeDef::Init*  
PCD required parameters
- *\_IO uint8\_t PCD\_HandleTypeDef::USB\_Address*  
USB Address
- *PCD\_EPTTypeDef PCD\_HandleTypeDef::IN\_ep[8]*  
IN endpoint parameters
- *PCD\_EPTTypeDef PCD\_HandleTypeDef::OUT\_ep[8]*  
OUT endpoint parameters
- *HAL\_LockTypeDef PCD\_HandleTypeDef::Lock*  
PCD peripheral status
- *\_IO PCD\_StateTypeDef PCD\_HandleTypeDef::State*  
PCD communication state
- *uint32\_t PCD\_HandleTypeDef::Setup[12]*  
Setup packet buffer
- *void\* PCD\_HandleTypeDef::pData*  
Pointer to upper stack Handler

## 32.2 PCD Firmware driver API description

### 32.2.1 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
    - \_\_HAL\_RCC\_USB\_CLK\_ENABLE();
  - 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();

### 32.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- *HAL\_PCD\_Init()*
- *HAL\_PCD\_DelInit()*
- *HAL\_PCD\_MspInit()*
- *HAL\_PCD\_MspDelInit()*

### 32.2.3 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()*

### 32.2.4 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()*
- *PCD\_WritePMA()*
- *PCD\_ReadPMA()*

### 32.2.5 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\\_PCD\\_GetState\(\)\*\*](#)

### 32.2.6 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\_DelInit**

Function Name      **HAL\_StatusTypeDef HAL\_PCD\_DelInit (PCD\_HandleTypeDef \* hpcd)**

Function Description      Delinitializes 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\_MspDelInit**

Function Name      **void HAL\_PCD\_MspDelInit (PCD\_HandleTypeDef \* hpcd)**

Function Description      Delinitializes 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 OTG Device.

Parameters      • **hpcd:** PCD handle

Return values      • **HAL:** status

**HAL\_PCD\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)</b>
Function Description	Stop The USB OTG Device.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_PCD\_IRQHandler**

Function Name	<b>void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)</b>
Function Description	This function handles PCD interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_PCD\_DataOutStageCallback**

Function Name	<b>void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Data out stage callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>epnum:</b> endpoint number</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_PCD\_DataInStageCallback**

Function Name	<b>void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Data IN stage callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>epnum:</b> endpoint number</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_PCD\_SetupStageCallback**

Function Name	<b>void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Setup stage callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_PCD\_SOFCallback**

Function Name	<b>void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	USB Start Of Frame callbacks.

---

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

### **HAL\_PCD\_ResetCallback**

Function Name	<b>void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### **HAL\_PCD\_SuspendCallback**

Function Name	<b>void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### **HAL\_PCD\_ResumeCallback**

Function Name	<b>void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Resume event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### **HAL\_PCD\_ISOOUTIncompleteCallback**

Function Name	<b>void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epi)</b>
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li><li>• <b>epi:</b> endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### **HAL\_PCD\_ISOINIncompleteCallback**

Function Name	<b>void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epi)</b>
Function Description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li><li>• <b>epi:</b> endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

**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	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> <li>• <b>ep_mps:</b> endpoint max packet size</li> <li>• <b>ep_type:</b> endpoint type</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_PCD\_EP\_Close

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Deactivate an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_PCD\_EP\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)</b>
Function Description	Receive an amount of data.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> <li>• <b>pBuf:</b> pointer to the reception buffer</li> <li>• <b>len:</b> amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_PCD\_EP\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)</b>
Function Description	Send an amount of data.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> <li>• <b>pBuf:</b> pointer to the transmission buffer</li> <li>• <b>len:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_PCD\_EP\_GetRxCount

Function Name	<b>uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Get Received Data Size.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• <b>ep_addr:</b> endpoint address</li> <li>• <b>Data:</b> Size</li> </ul>
---------------	---

### **HAL\_PCD\_EP\_SetStall**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Set a STALL condition over an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_PCD\_EP\_ClrStall**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Clear a STALL condition over in an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_PCD\_EP\_Flush**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Flush an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>ep_addr:</b> endpoint address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_PCD\_ActivateRemoteWakeup**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)</b>
Function Description	HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>status:</b></li> </ul>

### **HAL\_PCD\_DeActivateRemoteWakeup**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)</b>
Function Description	HAL_PCD_DeActivateRemoteWakeup : de-active remote wakeup signalling.

Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>status:</b></li> </ul>

### PCD\_WritePMA

Function Name	<b>void PCD_WritePMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)</b>
Function Description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	<ul style="list-style-type: none"> <li>• <b>USBx:</b> USB device</li> <li>• <b>pbUsrBuf:</b> pointer to user memory area.</li> <li>• <b>wPMABufAddr:</b> address into PMA.</li> <li>• <b>wNBytes:</b> no. of bytes to be copied.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### PCD\_ReadPMA

Function Name	<b>void PCD_ReadPMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)</b>
Function Description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	<ul style="list-style-type: none"> <li>• <b>USBx:</b> USB device</li> <li>• <b>pbUsrBuf:</b> pointer to user memory area.</li> <li>• <b>wPMABufAddr:</b> address into PMA.</li> <li>• <b>wNBytes:</b> no. of bytes to be copied.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_PCD\_GetState

Function Name	<b>PCD_HandleTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)</b>
Function Description	Return the PCD state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

## 32.3 PCD Firmware driver defines

### 32.3.1 PCD

#### *PCD End Point*

PCD\_ENDP0  
 PCD\_ENDP1  
 PCD\_ENDP2  
 PCD\_ENDP3  
 PCD\_ENDP4

`PCD_ENDP5``PCD_ENDP6``PCD_ENDP7``PCD_SNG_BUF``PCD_DBL_BUF``IS_PCD_ALL_INSTANCE`***PCD Interrupt***`__HAL_PCD_GET_FLAG``__HAL_PCD_CLEAR_FLAG``USB_WAKEUP_EXTI_LINE`

External interrupt line 18 Connected to the USB FS EXTI Line

`__HAL_USB_WAKEUP_EXTI_ENABLE_IT``__HAL_USB_WAKEUP_EXTI_DISABLE_IT``__HAL_USB_EXTI_GENERATE_SWIT``PCD_SET_ENDPOINT``PCD_GET_ENDPOINT``PCD_SET_EPTYPE`**Description:**

- sets the type in the endpoint register(bits EP\_TYPE[1:0])

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wType: Endpoint Type.

**Return value:**

- None

`PCD_GET_EPTYPE`**Description:**

- gets the type in the endpoint register(bits EP\_TYPE[1:0])

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- Endpoint: Type

`PCD_FreeUserBuffer`**Description:**

- free buffer used from the application realizing it to the line toggles bit SW\_BUF in the double buffered endpoint register

**Parameters:**

- USBx: USB device.
- bEpNum

**Return value:**

- None

[PCD\\_GET\\_DB\\_DIR](#)

**Description:**

- gets direction of the double buffered endpoint

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- EP\_DBUF\_OUT: if the endpoint counter not yet programmed.

[PCD\\_SET\\_EP\\_TX\\_STATUS](#)

**Description:**

- sets the status for tx transfer (bits STAT\_TX[1:0]).

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wState: new state

**Return value:**

- None

[PCD\\_SET\\_EP\\_RX\\_STATUS](#)

**Description:**

- sets the status for rx transfer (bits STAT\_RX[1:0])

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wState: new state

**Return value:**

- None

[PCD\\_SET\\_EP\\_TXRX\\_STATUS](#)

**Description:**

- sets the status for rx & tx (bits STAT\_RX[1:0] & STAT\_TX[1:0])

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

**Return value:**

PCD\_GET\_EP\_TX\_STATUS

- None

**Description:**

- gets the status for tx/rx transfer (bits STAT\_TX[1:0] /STAT\_RX[1:0])

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- status

PCD\_GET\_EP\_RX\_STATUS

**Description:**

- sets directly the VALID tx/rx-status into the endpoint register

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_SET\_EP\_RX\_VALID

**Description:**

- checks stall condition in an endpoint.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- TRUE: = endpoint in stall condition.

PCD\_GET\_EP\_RX\_STALL\_STATUS

**Description:**

- set & clear EP\_KIND bit.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_CLEAR\_EP\_KIND

**Description:**

- Sets/clears directly STATUS\_OUT bit in the endpoint register.

PCD\_CLEAR\_OUT\_STATUS  
PCD\_SET\_EP\_DBUF

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

**Description:**

- Sets/clears directly EP\_KIND bit in the endpoint register.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_CLEAR\_EP\_DBUF  
PCD\_CLEAR\_RX\_EP\_CTR

**Description:**

- Clears bit CTR\_RX / CTR\_TX in the endpoint register.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_CLEAR\_TX\_EP\_CTR  
PCD\_RX\_DTOG

**Description:**

- Toggles DTOG\_RX / DTOG\_TX bit in the endpoint register.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_TX\_DTOG  
PCD\_CLEAR\_RX\_DTOG

**Description:**

- Clears DTOG\_RX / DTOG\_TX bit in the endpoint register.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

	<b>Return value:</b>
PCD_CLEAR_TX_DTOG	<ul style="list-style-type: none"><li>• None</li></ul>
PCD_SET_EP_ADDRESS	<b>Description:</b> <ul style="list-style-type: none"><li>• Sets address in an endpoint register.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• USBx: USB device.</li><li>• bEpNum: Endpoint Number.</li><li>• bAddr: Address.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None</li></ul>
PCD_GET_EP_ADDRESS	<b>Description:</b> <ul style="list-style-type: none"><li>• Gets address in an endpoint register.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• USBx: USB device.</li><li>• bEpNum: Endpoint Number.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None</li></ul>
PCD_EP_TX_ADDRESS	
PCD_EP_TX_CNT	
PCD_EP_RX_ADDRESS	
PCD_EP_RX_CNT	
PCD_SET_EP_TX_ADDRESS	<b>Description:</b> <ul style="list-style-type: none"><li>• sets address of the tx/rx buffer.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• USBx: USB device.</li><li>• bEpNum: Endpoint Number.</li><li>• wAddr: address to be set (must be word aligned).</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None</li></ul>
PCD_SET_EP_RX_ADDRESS	
PCD_GET_EP_TX_ADDRESS	<b>Description:</b> <ul style="list-style-type: none"><li>• Gets address of the tx/rx buffer.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• USBx: USB device.</li><li>• bEpNum: Endpoint Number.</li></ul> <b>Return value:</b>

PCD\_GET\_EP\_RX\_ADDRESS

- address: of the buffer.

PCD\_CALC\_BLK32

**Description:**

- Sets counter of rx buffer with no.

**Parameters:**

- dwReg: Register.
- wCount: Counter.
- wNBlocks: Nb of block

**Return value:**

- None

PCD\_CALC\_BLK2

PCD\_SET\_EP\_CNT\_RX\_REG

PCD\_SET\_EP\_RX\_DBUF0\_CNT

PCD\_SET\_EP\_TX\_CNT

**Description:**

- sets counter for the tx/rx buffer.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wCount: Counter value.

**Return value:**

- None

PCD\_SET\_EP\_RX\_CNT

**Description:**

- gets counter of the tx buffer.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- Counter: value

PCD\_GET\_EP\_RX\_CNT

PCD\_SET\_EP\_DBUF0\_ADDR

**Description:**

- Sets buffer 0/1 address in a double buffer endpoint.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

**Return value:**

PCD\_SET\_EP\_DBUF1\_ADDR

- Counter: value

PCD\_SET\_EP\_DBUF\_ADDR

**Description:**

- Sets addresses in a double buffer endpoint.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.
- wBuf1Addr: = buffer 1 address.

**Return value:**

- None

PCD\_GET\_EP\_DBUF0\_ADDR

**Description:**

- Gets buffer 0/1 address of a double buffer endpoint.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_GET\_EP\_DBUF1\_ADDR

**Description:**

- Gets buffer 0/1 address of a double buffer endpoint.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.
- bDir: endpoint dir EP\_DBUF\_OUT = OUT and EP\_DBUF\_IN = IN
- wCount: Counter value

**Return value:**

- None

PCD\_SET\_EP\_DBUF1\_CNT

**Description:**

PCD\_SET\_EP\_DBUF\_CNT

**Description:**

PCD\_GET\_EP\_DBUF0\_CNT

- Gets buffer 0/1 rx/tx counter for double buffering.

**Parameters:**

- USBx: USB device.
- bEpNum: Endpoint Number.

**Return value:**

- None

PCD\_GET\_EP\_DBUF1\_CNT

***PCD Speed***

PCD\_SPEED\_HIGH

PCD\_SPEED\_FULL

***PCD USB Core PHY***

PCD\_PHY\_EMBEDDED

***PCD USB EP0 MPS***

DEP0CTL\_MPS\_64

DEP0CTL\_MPS\_32

DEP0CTL\_MPS\_16

DEP0CTL\_MPS\_8

PCD\_EP0MPS\_64

PCD\_EP0MPS\_32

PCD\_EP0MPS\_16

PCD\_EP0MPS\_08

***PCD USB EP Type***

PCD\_EP\_TYPE\_CTRL

PCD\_EP\_TYPE\_ISOC

PCD\_EP\_TYPE\_BULK

PCD\_EP\_TYPE\_INTR

## 33 HAL PCD Extension Driver

### 33.1 PCDEEx Firmware driver API description

#### 33.1.1 Peripheral extended features functions

This section contains the following APIs:

- [\*HAL\\_PCDEEx\\_PMACConfig\(\)\*](#)

#### 33.1.2 Detailed description of functions

##### **HAL\_PCDEEx\_PMACConfig**

Function Name	<b>HAL_StatusTypeDef HAL_PCDEEx_PMACConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaaddress)</b>
Function Description	Configure PMA for EP.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> : Device instance</li><li>• <b>ep_addr:</b> endpoint address</li><li>• <b>ep_kind:</b> endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used</li><li>• <b>pmaaddress:</b> 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.</li></ul>
Return values	<ul style="list-style-type: none"><li>• :: status</li></ul>

## 34 HAL PWR Generic Driver

### 34.1 PWR Firmware driver registers structures

#### 34.1.1 PWR\_PVDTTypeDef

##### Data Fields

- *uint32\_t PVDLevel*
- *uint32\_t Mode*

##### Field Documentation

- *uint32\_t PWR\_PVDTTypeDef::PVDLevel*  
PVDLevel: Specifies the PVD detection level. This parameter can be a value of [\*PWR\\_PVD\\_detection\\_level\*](#)
- *uint32\_t PWR\_PVDTTypeDef::Mode*  
Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [\*PWR\\_PVD\\_Mode\*](#)

### 34.2 PWR Firmware driver API description

#### 34.2.1 Initialization and de-initialization functions

This section contains the following APIs:

- [\*HAL\\_PWR\\_DelInit\(\)\*](#)
- [\*HAL\\_PWR\\_EnableBkUpAccess\(\)\*](#)
- [\*HAL\\_PWR\\_DisableBkUpAccess\(\)\*](#)

#### 34.2.2 Peripheral Control functions

##### Backup domain

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.

##### 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).
- The PVD can use an external input analog voltage (PVD\_IN) which is compared internally to VREFINT. The PVD\_IN (PB7) has to be configured in Analog mode when PWR\_PVDLevel\_7 is selected (PLS[2:0] = 111).

- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through \_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_IT() macro.
- The PVD is stopped in Standby mode.

### 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 are two WakeUp pins: WakeUp Pin 1 on PA.00. WakeUp Pin 2 on PC.13. WakeUp Pin 3 on PE.06 .

### Main and Backup Regulators configuration

#### Low Power modes configuration

The device features 5 low-power modes:

- Low power run mode: regulator in low power mode, limited clock frequency, limited number of peripherals running.
- Sleep mode: Cortex-M0+ core stopped, peripherals kept running.
- Low power sleep mode: Cortex-M0+ core stopped, limited clock frequency, limited number of peripherals running, regulator in low power mode.
- Stop mode: All clocks are stopped, regulator running, regulator in low power mode.
- Standby mode: VCORE domain powered off

#### Low power run mode

To further reduce the consumption when the system is in Run mode, the regulator can be configured in low power mode. In this mode, the system frequency should not exceed MSI frequency range1. In Low power run mode, all I/O pins keep the same state as in Run mode.

- Entry:
  - VCORE in range2
  - Decrease the system frequency not to exceed the frequency of MSI frequency range1.
  - The regulator is forced in low power mode using the HAL\_PWREx\_EnableLowPowerRunMode() function.
- Exit:
  - The regulator is forced in Main regulator mode using the HAL\_PWREx\_DisableLowPowerRunMode() function.
  - Increase the system frequency if needed.

#### 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:
  - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode. If the WFE instruction was used to enter sleep mode, the MCU exits Sleep mode as soon as an event occurs.

### Low power sleep mode

- Entry: The Low power sleep mode is entered by using the HAL\_PWR\_EnterSLEEPMode(PWR\_LOWPOWERREGULATOR\_ON, PWR\_SLEEPENTRY\_WFx) functions with
  - PWR\_SLEEPENTRY\_WFI: enter SLEEP mode with WFI instruction
  - PWR\_SLEEPENTRY\_WFE: enter SLEEP mode with WFE instruction
- The Flash memory can be switched off by using the control bits (SLEEP\_PD in the FLASH\_ACR register. This reduces power consumption but increases the wake-up time.
- Exit:
  - If the WFI instruction was used to enter Low power sleep mode, any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Low power sleep mode. If the WFE instruction was used to enter Low power sleep mode, the MCU exits Sleep mode as soon as an event occurs.

### Stop mode

The Stop mode is based on the Cortex-M0+ 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 VCORE domain are stopped, the PLL, the MSI, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. To get the lowest consumption in Stop mode, the internal Flash memory also enters low power mode. When the Flash memory is in power-down mode, an additional startup delay is incurred when waking up from Stop mode. To minimize the consumption In Stop mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering Stop mode. They can be switched on again by software after exiting Stop mode using the ULP bit in the PWR\_CR register. 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 function with:
  - Main regulator ON.
  - Low Power regulator ON.
  - PWR\_SLEEPENTRY\_WFI: enter SLEEP mode with WFI instruction
  - PWR\_SLEEPENTRY\_WFE: enter SLEEP mode with WFE instruction
- Exit:
  - By issuing an interrupt or a wakeup event, the MSI or HSI16 RC oscillator is selected as system clock depending the bit STOPWUCK in the RCC\_CFGR register

### Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M0+ deepsleep mode, with the voltage regulator disabled. The VCORE domain is consequently powered off. The PLL, the MSI, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry. To minimize the consumption In Standby mode,

VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering the Standby mode. They can be switched on again by software after exiting the Standby mode. function.

- Entry:
  - The Standby mode is entered using the HAL\_PWR\_EnterSTANDBYMode() function.
- Exit:
  - WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

### Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop mode
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to:
    - Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI\_Init() function.
    - Enable the RTC Alarm Interrupt using the RTC\_ITConfig() function
    - Configure the RTC to generate the RTC alarm using the RTC\_SetAlarm() and RTC\_AlarmCmd() functions.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to:
    - Configure the EXTI Line 19 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI\_Init() function.
    - Enable the RTC Tamper or time stamp Interrupt using the RTC\_ITConfig() function.
    - Configure the RTC to detect the tamper or time stamp event using the RTC\_TimeStampConfig(), RTC\_TamperTriggerConfig() and RTC\_TamperCmd() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to:
    - Configure the EXTI Line 20 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI\_Init() function.
    - Enable the RTC WakeUp Interrupt using the RTC\_ITConfig() function.
    - Configure the RTC to generate the RTC WakeUp event using the RTC\_WakeUpClockConfig(), RTC\_SetWakeUpCounter() and RTC\_WakeUpCmd() functions.
- RTC auto-wakeup (AWU) from the Standby mode
  - To wake up from the Standby mode with an RTC alarm event, it is necessary to:
    - Enable the RTC Alarm Interrupt using the RTC\_ITConfig() function.
    - Configure the RTC to generate the RTC alarm using the RTC\_SetAlarm() and RTC\_AlarmCmd() functions.
  - To wake up from the Standby mode with an RTC Tamper or time stamp event, it is necessary to:
    - Enable the RTC Tamper or time stamp Interrupt using the RTC\_ITConfig() function.
    - Configure the RTC to detect the tamper or time stamp event using the RTC\_TimeStampConfig(), RTC\_TamperTriggerConfig() and RTC\_TamperCmd() functions.
  - To wake up from the Standby mode with an RTC WakeUp event, it is necessary to:
    - Enable the RTC WakeUp Interrupt using the RTC\_ITConfig() function

- Configure the RTC to generate the RTC WakeUp event using the RTC\_WakeUpClockConfig(), RTC\_SetWakeUpCounter() and RTC\_WakeUpCmd() functions.
- Comparator auto-wakeup (AWU) from the Stop mode
  - To wake up from the Stop mode with an comparator 1 or comparator 2 wakeup event, it is necessary to:
    - Configure the EXTI Line 21 for comparator 1 or EXTI Line 22 for comparator 2 to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the EXTI\_Init() function.
    - Configure the comparator to generate the event.

This section contains the following APIs:

- [\*\*\*HAL\\_PWR\\_EnableBkUpAccess\(\)\*\*\*](#)
- [\*\*\*HAL\\_PWR\\_DisableBkUpAccess\(\)\*\*\*](#)
- [\*\*\*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\(\)\*\*\*](#)

### 34.2.3 Detailed description of functions

#### **HAL\_PWR\_DeInit**

Function Name	<b>void HAL_PWR_DeInit (void )</b>
Function Description	Deinitializes the HAL PWR peripheral registers to their default reset values.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### **HAL\_PWR\_EnableBkUpAccess**

Function Name	<b>void HAL_PWR_EnableBkUpAccess (void )</b>
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers ).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li> </ul>

#### **HAL\_PWR\_DisableBkUpAccess**

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

Function Description	Disables access to the backup domain.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Applies to RTC registers, RTC backup data registers.</li> <li>• If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li> </ul>

### HAL\_PWR\_ConfigPVD

Function Name	<b>void HAL_PWR_ConfigPVD (PWR_PVDTTypeDef * sConfigPVD)</b>
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> <li>• <b>sConfigPVD:</b> pointer to an PWR_PVDTTypeDef structure that contains the configuration information for the PVD.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.</li> </ul>

### HAL\_PWR\_EnablePVD

Function Name	<b>void HAL_PWR_EnablePVD (void )</b>
Function Description	Enables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_PWR\_DisablePVD

Function Name	<b>void HAL_PWR_DisablePVD (void )</b>
Function Description	Disables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_PWR\_PVD\_IRQHandler

Function Name	<b>void HAL_PWR_PVD_IRQHandler (void )</b>
Function Description	This function handles the PWR PVD interrupt request.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be called under the PVD_IRQHandler().</li> </ul>

### HAL\_PWR\_PVDCallback

Function Name	<b>void HAL_PWR_PVDCallback (void )</b>
Function Description	PWR PVD interrupt callback.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_PWR\_EnableWakeUpPin**

Function Name	<b>void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)</b>
Function Description	Enables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPinx:</b> Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– PWR_WAKEUP_PIN1</li> <li>– PWR_WAKEUP_PIN2</li> <li>– PWR_WAKEUP_PIN3 for stm32l07xxx and stm32l08xxx devices only.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_PWR\_DisableWakeUpPin**

Function Name	<b>void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)</b>
Function Description	Disables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPinx:</b> Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– PWR_WAKEUP_PIN1</li> <li>– PWR_WAKEUP_PIN2</li> <li>– PWR_WAKEUP_PIN3 for stm32l07xxx and stm32l08xxx devices only.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_PWR\_EnterSTOPMode**

Function Name	<b>void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)</b>
Function Description	Enters Stop mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Regulator:</b> Specifies the regulator state in Stop mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– PWR_MAINREGULATOR_ON: Stop mode with regulator ON</li> <li>– PWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON</li> </ul> </li> <li>• <b>STOPEntry:</b> Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction</li> <li>– PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Stop mode, all I/O pins keep the same state as in Run mode.</li> <li>• When exiting Stop mode by issuing an interrupt or a wakeup event, MSI or HSI16 RC oscillator is selected as system clock depending the bit STOPWUCK in the RCC_CFGR register.</li> <li>• When the voltage regulator operates in low power mode, an</li> </ul>

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.

- Before entering in this function, it is important to ensure that the WUF wakeup flag is cleared. To perform this action, it is possible to call the following macro :  
`_HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU)`

### **HAL\_PWR\_EnterSLEEPMode**

Function Name	<code>void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)</code>
Function Description	Enters Sleep mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Regulator:</b> Specifies the regulator state in SLEEP mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- PWR_MAINREGULATOR_ON: SLEEP mode with regulator ON</li> <li>- PWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON</li> </ul> </li> <li>• <b>SLEEPEntry:</b> 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: <ul style="list-style-type: none"> <li>- PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction</li> <li>- PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Sleep mode, all I/O pins keep the same state as in Run mode.</li> </ul>

### **HAL\_PWR\_EnterSTANDBYMode**

Function Name	<code>void HAL_PWR_EnterSTANDBYMode (void )</code>
Function Description	Enters Standby mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)RTC_AF1 pin (PC13) if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out.RTC_AF2 pin (PC13) if configured for tamper.WKUP pin 1 (PA00) if enabled.WKUP pin 2 (PC13) if enabled.WKUP pin 3 (PE06) if enabled, for stm32l07xxx and stm32l08xxx devices only.WKUP pin 3 (PA02) if enabled, for stm32l031xx devices only.</li> </ul>

### **HAL\_PWR\_EnableSleepOnExit**

Function Name	<code>void HAL_PWR_EnableSleepOnExit (void )</code>
Function Description	Indicates Sleep-On-Exit when returning from Handler mode to

Thread mode.

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.</li> </ul>

### **HAL\_PWR\_DisableSleepOnExit**

Function Name	<b>void HAL_PWR_DisableSleepOnExit (void )</b>
Function Description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.</li> </ul>

### **HAL\_PWR\_EnableSEVOnPend**

Function Name	<b>void HAL_PWR_EnableSEVOnPend (void )</b>
Function Description	Enables CORTEX M0+ SEVONPEND bit.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.</li> </ul>

### **HAL\_PWR\_DisableSEVOnPend**

Function Name	<b>void HAL_PWR_DisableSEVOnPend (void )</b>
Function Description	Disables CORTEX M0+ SEVONPEND bit.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.</li> </ul>

## **34.3 PWR Firmware driver defines**

### **34.3.1 PWR**

#### **PWR Exported Macros**

<b>__HAL_PWR_VOLTAGESCALING_CONFIG</b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>macros configure the main internal regulator output voltage.</li> </ul>
<b>Parameters:</b>	
	<ul style="list-style-type: none"> <li><b>__REGULATOR__:</b> specifies the regulator output voltage to achieve a</li> </ul>

tradeoff between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheets for more details). This parameter can be one of the following values:

- PWR\_REGULATOR\_VOLTAGE\_E\_SCALE1: Regulator voltage output Scale 1 mode, System frequency up to 32 MHz.
- PWR\_REGULATOR\_VOLTAGE\_E\_SCALE2: Regulator voltage output Scale 2 mode, System frequency up to 16 MHz.
- PWR\_REGULATOR\_VOLTAGE\_E\_SCALE3: Regulator voltage output Scale 3 mode, System frequency up to 4.2 MHz

**Return value:**

- None

**\_HAL\_PWR\_GET\_FLAG**

- Check PWR flag is set or not.

**Parameters:**

- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - PWR\_FLAG\_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
  - PWR\_FLAG\_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  - PWR\_FLAG\_PVDO: PVD Output. This flag is valid only if PVD is enabled by the HAL\_PWR\_EnablePVD() function. The PVD is stopped by Standby mode. For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.
  - PWR\_FLAG\_VREFINTRDY:

Internal voltage reference (VREFINT) ready flag. This bit indicates the state of the internal voltage reference, VREFINT.

- PWR\_FLAG\_VOS: Voltage Scaling select flag. A delay is required for the internal regulator to be ready after the voltage range is changed. The VOSF bit indicates that the regulator has reached the voltage level defined with bits VOS of PWR\_CR register.
- PWR\_FLAG\_REGLP: Regulator LP flag. When the MCU exits from Low power run mode, this bit stays at 1 until the regulator is ready in main mode. A polling on this bit is recommended to wait for the regulator main mode. This bit is reset by hardware when the regulator is ready.

#### **Return value:**

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

### **\_\_HAL\_PWR\_CLEAR\_FLAG**

#### **Description:**

- Clear the PWR 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`

- 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.

`_HAL_PWR_PVD_EXTI_GENERATE_SWIT`**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.

***PWREx Flag Setting Time Out Value***`PWR_FLAG_SETTING_DELAY_US`***PWR Flag***`PWR_FLAG_WU``PWR_FLAG_SB``PWR_FLAG_PVDO``PWR_FLAG_VREFINTRDY``PWR_FLAG_VOS``PWR_FLAG_REGLP`***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\_WAKEUP\_PIN1

PWR\_WAKEUP\_PIN2

PWR\_WAKEUP\_PIN3

**PWR Regulator state in SLEEP/STOP mode**

PWR\_MAINREGULATOR\_ON

PWR\_LOWPOWERREGULATOR\_ON

**PWR Regulator Voltage Scale**

PWR\_REGULATOR\_VOLTAGE\_SCALE1

PWR\_REGULATOR\_VOLTAGE\_SCALE2

PWR\_REGULATOR\_VOLTAGE\_SCALE3

IS\_PWR\_VOLTAGE\_SCALING\_RANGE

**PWR SLEEP mode entry**

PWR\_SLEEPENTRY\_WFI

PWR\_SLEEPENTRY\_WFE

**PWR STOP mode entry**

PWR\_STOPENTRY\_WFI

PWR\_STOPENTRY\_WFE

## 35 HAL PWR Extension Driver

### 35.1 PWREx Firmware driver defines

#### 35.1.1 PWREx

##### *PWREx Exported Macros*

`_HAL_PWR_FLASHWAKEUP_ENABLE`

##### **Notes:**

- When entering low power mode (stop or standby only), if DS\_EE\_KOFF and RUN\_PD of FLASH\_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.

`_HAL_PWR_FLASHWAKEUP_DISABLE`

##### **Notes:**

- When entering low power mode (stop or standby only), if DS\_EE\_KOFF and RUN\_PD of FLASH\_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.

## 36 HAL RCC Generic Driver

### 36.1 RCC Firmware driver registers structures

#### 36.1.1 RCC\_PLLInitTypeDef

##### Data Fields

- *uint32\_t PLLState*
- *uint32\_t PLLSource*
- *uint32\_t PLLMUL*
- *uint32\_t PLLDIV*

##### Field Documentation

- *uint32\_t RCC\_PLLInitTypeDef::PLLState*  
The new state of the PLL. This parameter can be a value of [RCC\\_PLL\\_Config](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLSource*  
RCC\_PLLSource: PLL entry clock source. This parameter must be a value of [RCC\\_PLL\\_Clock\\_Source](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLMUL*  
PLLMUL: Multiplication factor for PLL VCO output clock This parameter must be a value of [RCC\\_PLLMultiplication\\_Factor](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLDIV*  
PLLDIV: Division factor for main system clock (SYSCLK) This parameter must be a value of [RCC\\_PLLDivider\\_Factor](#)

#### 36.1.2 RCC\_OscInitTypeDef

##### Data Fields

- *uint32\_t OscillatorType*
- *uint32\_t HSEState*
- *uint32\_t LSEState*
- *uint32\_t HSISState*
- *uint32\_t HSICalibrationValue*
- *uint32\_t LSISState*
- *uint32\_t HSI48State*
- *uint32\_t MSISState*
- *uint32\_t MSICalibrationValue*
- *uint32\_t MSIClockRange*
- *RCC\_PLLInitTypeDef PLL*

### Field Documentation

- **`uint32_t RCC_OscInitTypeDef::OscillatorType`**  
The oscillators to be configured. This parameter can be a value of **`RCC_Oscillator_Type`**
- **`uint32_t RCC_OscInitTypeDef::HSEState`**  
The new state of the HSE. This parameter can be a value of **`RCC_HSE_Config`**
- **`uint32_t RCC_OscInitTypeDef::LSEState`**  
The new state of the LSE. This parameter can be a value of **`RCC_LSE_Config`**
- **`uint32_t RCC_OscInitTypeDef::HSIState`**  
The new state of the HSI. This parameter can be a value of **`RCC_HSI_Config`**
- **`uint32_t RCC_OscInitTypeDef::HSICalibrationValue`**  
The calibration trimming value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- **`uint32_t RCC_OscInitTypeDef::LSIState`**  
The new state of the LSI. This parameter can be a value of **`RCC_LSI_Config`**
- **`uint32_t RCC_OscInitTypeDef::HSI48State`**  
The new state of the HSI48. This parameter can be a value of **`RCC_HSI48_Config`**
- **`uint32_t RCC_OscInitTypeDef::MSIState`**  
The new state of the MSI. This parameter can be a value of **`RCC_MSI_Config`**
- **`uint32_t RCC_OscInitTypeDef::MSICalibrationValue`**  
The calibration trimming value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF
- **`uint32_t RCC_OscInitTypeDef::MSIClockRange`**  
The MSI frequency range. This parameter can be a value of **`RCC_MSI_Clock_Range`**
- **`RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL`**  
PLL structure parameters

### 36.1.3 `RCC_ClkInitTypeDef`

#### Data Fields

- **`uint32_t ClockType`**
- **`uint32_t SYSCLKSource`**
- **`uint32_t AHBCLKDivider`**
- **`uint32_t APB1CLKDivider`**
- **`uint32_t APB2CLKDivider`**

#### Field Documentation

- **`uint32_t RCC_ClkInitTypeDef::ClockType`**  
The clock to be configured. This parameter can be a value of **`RCC_System_Clock_Type`**
- **`uint32_t RCC_ClkInitTypeDef::SYSCLKSource`**  
The clock source (SYSCLK) used as system clock. This parameter can be a value of **`RCC_System_Clock_Source`**
- **`uint32_t RCC_ClkInitTypeDef::AHBCLKDivider`**  
The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of **`RCC_AHB_Clock_Source`**

- **`uint32_t RCC_ClkInitTypeDef::APB1CLKDivider`**  
The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK).  
This parameter can be a value of [RCC\\_APB1\\_APB2\\_Clock\\_Source](#)
- **`uint32_t RCC_ClkInitTypeDef::APB2CLKDivider`**  
The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK).  
This parameter can be a value of [RCC\\_APB1\\_APB2\\_Clock\\_Source](#)

## 36.2 RCC Firmware driver API description

### 36.2.1 RCC specific features

After reset the device is running from MSI (2 MHz) with Flash 0 WS, all peripherals are off except internal SRAM, Flash and SW-DP.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at MSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the SW-DP pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (ADC, RTC/LCD, RNG and IWDG)

### 36.2.2 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 registeres.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Possible Workarounds:

1. Enable the peripheral clock sometimes before the peripheral read/write register is required.
2. For AHB peripheral, insert two dummy read to the peripheral register.
3. For APB peripheral, insert a dummy read to the peripheral register.

### 36.2.3 Initialization and de-initialization functions

This section provide functions allowing to configure the internal/external clocks, PLL, CSS and MCO.

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.

2. MSI (multi-speed internal), multispeed low power RC (65.536 KHz to 4.194 MHz) MHz used as System clock source.
3. LSI (low-speed internal), 37 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 1 to 24 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL (clocked by HSI or HSE), for System clock and USB (48 MHz).
7. CSS (Clock security system), once enable and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to MSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
8. MCO (microcontroller clock output), used to output SYSCLK, HSI, MSI, HSE, PLL, LSI or LSE clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): MSI, 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 IOPORT, AHB bus (DMA,Flash...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL\_RCC\_GetSysClockFreq()" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: I2S: the I2S clock can be derived from an external clock mapped on the I2S\_CKIN pin. RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use \_\_HAL\_RCC\_RTC\_CONFIG() and \_\_HAL\_RCC\_RTC\_ENABLE() macros to configure this clock. USB FS, and RNG require a frequency equal to 48 MHz to work correctly. This clock is derived from the main PLL or HSI48 RC oscillator. IWDG clock which is always the LSI clock.
2. For the STM32L0xx devices, the maximum frequency of the SYSCLK ,HCLK, APB1 and APB2 is 32 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly. Refer to the Reference Manual for more details.

This section contains the following APIs:

- [\*\*\*HAL\\_RCC\\_DeInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_OscConfig\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_ClockConfig\(\)\*\*\*](#)

### 36.2.4 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\\_GetSysClockFreq\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_GetHCLKFreq\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_GetPCLK1Freq\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_GetPCLK2Freq\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_GetOscConfig\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_GetClockConfig\(\)\*\*\*](#)
- [\*\*\*HAL\\_RCC\\_NMI\\_IRQHandler\(\)\*\*\*](#)

- [\*\*HAL\\_RCC\\_CSSCallback\(\)\*\*](#)

### 36.2.5 Detailed description of functions

#### **HAL\_RCC\_DelInit**

Function Name	<b>void HAL_RCC_DelInit (void )</b>
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The default reset state of the clock configuration is given below: MSI ON and used as system clock source (MSI range is not modified by this function, it keep the value configured by user application) HSI, HSI_OUT, HSE and PLL OFF AHB, APB1 and APB2 prescaler set to 1. CSS and MCO OFF All interrupts disabled</li> <li>• This function does not modify the configuration of the <ul style="list-style-type: none"> <li>-Peripheral clocks</li> <li>-HSI48, LSI, LSE and RTC clocks</li> </ul> </li> </ul>

#### **HAL\_RCC\_OscConfig**

Function Name	<b>HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</b>
Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_OscInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The PLL is not disabled when used as system clock.</li> <li>• 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.</li> <li>• 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.</li> </ul>

#### **HAL\_RCC\_ClockConfig**

Function Name	<b>HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)</b>
Function Description	Initializes the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct:</b> pointer to an RCC_ClkInitTypeDef structure that contains the configuration information for the RCC peripheral.</li> <li>• <b>FLatency:</b> FLASH Latency, this parameter depends on System Clock Frequency</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- |       |  |
|-------|--|
| Notes | <ul style="list-style-type: none"> <li>• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function</li> <li>• The MSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).</li> <li>• A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready.</li> </ul> |
|-------|--|

### **HAL\_RCC\_MCOConfig**

Function Name	<b>void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOsource, uint32_t RCC_MCODiv)</b>
Function Description	Selects the clock source to output on MCO pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_MCOx:</b> specifies the output direction for the clock source. For STM32L0xx family this parameter can have only one value: <ul style="list-style-type: none"> <li>– RCC_MCO1: Clock source to output on MCO pin(PA8).</li> <li>– RCC_MCO2: Clock source to output on MCO pin(PA9).</li> <li>– RCC_MCO3: Clock source to output on MCO pin(PB13) on STM32L03x/4x/7x/8x .</li> </ul> </li> <li>• <b>RCC_MCOsource:</b> specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RCC_MCO1SOURCE_NOCLOCK: No clock selected</li> <li>– RCC_MCO1SOURCE_SYSCLK: System clock selected</li> <li>– RCC_MCO1SOURCE_HSI: HSI oscillator clock selected</li> <li>– RCC_MCO1SOURCE_MS1: MSI oscillator clock selected</li> <li>– RCC_MCO1SOURCE_HSE: HSE oscillator clock selected</li> <li>– RCC_MCO1SOURCE_PLLCLK: PLL clock selected</li> <li>– RCC_MCO1SOURCE_LSI: LSI clock selected</li> <li>– RCC_MCO1SOURCE_LSE: LSE clock selected and in STM32L052xx,STM32L053xx,STM32L062xx, STM32L063xx</li> <li>– RCC_MCO1SOURCE_LSE: LSE clock selected and in STM32L072xx,STM32L073xx,STM32L082xx, STM32L083xx</li> <li>– RCC_MCO1SOURCE_HSI48: HSI48 clock selected</li> </ul> </li> <li>• <b>RCC_MCODiv:</b> specifies the MCO DIV. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RCC_MCODIV_1: no division applied to MCO clock</li> <li>– RCC_MCODIV_2: division by 2 applied to MCO clock</li> <li>– RCC_MCODIV_4: division by 4 applied to MCO clock</li> <li>– RCC_MCODIV_8: division by 8 applied to MCO clock</li> <li>– RCC_MCODIV_16: division by 16 applied to MCO clock</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- |       |  |
|-------|--|
| Notes | <ul style="list-style-type: none"> <li>MCO pin should be configured in alternate function mode.</li> </ul> |
|-------|--|

### **HAL\_RCC\_EnableCSS**

Function Name	<b>void HAL_RCC_EnableCSS (void )</b>
Function Description	Enables the Clock Security System.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M0+ NMI (Non-Maskable Interrupt) exception vector.</li> </ul>

### **HAL\_RCC\_GetSysClockFreq**

Function Name	<b>uint32_t HAL_RCC_GetSysClockFreq (void )</b>
Function Description	Returns the SYSCLK frequency.
Return values	<ul style="list-style-type: none"> <li><b>SYSCLK:</b> frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:</li> <li>If SYSCLK source is MSI, function returns values based on MSI Value as defined by the MSI range.</li> <li>If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)</li> <li>If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)</li> <li>If SYSCLK source is PLL, function returns values based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors.</li> <li>(*) HSI_VALUE is a constant defined in stm32l0xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.</li> <li>(**) HSE_VALUE is a constant defined in stm32l0xx_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.</li> <li>The result of this function could be not correct when using fractional value for HSE crystal.</li> <li>This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.</li> <li>Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.</li> </ul>

### **HAL\_RCC\_GetHCLKFreq**

Function Name	<b>uint32_t HAL_RCC_GetHCLKFreq (void )</b>
---------------	---

Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> <li><b>HCLK:</b> frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.</li> <li>The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function</li> </ul>

### **HAL\_RCC\_GetPCLK1Freq**

Function Name	<b>uint32_t HAL_RCC_GetPCLK1Freq (void )</b>
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> <li><b>PCLK1:</b> frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.</li> </ul>

### **HAL\_RCC\_GetPCLK2Freq**

Function Name	<b>uint32_t HAL_RCC_GetPCLK2Freq (void )</b>
Function Description	Returns the PCLK2 frequency.
Return values	<ul style="list-style-type: none"> <li><b>PCLK2:</b> frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.</li> </ul>

### **HAL\_RCC\_GetOscConfig**

Function Name	<b>void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</b>
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li><b>RCC_OscInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that will be configured.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### **HAL\_RCC\_GetClockConfig**

Function Name	<b>void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)</b>
Function Description	Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li><b>RCC_ClkInitStruct:</b> pointer to an RCC_ClkInitTypeDef structure that will be configured.</li> <li><b>pFLatency:</b> Pointer on the Flash Latency.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

**HAL\_RCC\_NMI\_IRQHandler**

Function Name	<b>void HAL_RCC_NMI_IRQHandler (void )</b>
Function Description	This function handles the RCC CSS interrupt request.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be called under the NMI_Handler().</li> </ul>

**HAL\_RCC\_CSSCallback**

Function Name	<b>void HAL_RCC_CSSCallback (void )</b>
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 36.3 RCC Firmware driver defines

### 36.3.1 RCC

***AHB Peripheral Clock Sleep Enable Disable***

```
_HAL_RCC_CRC_CLK_SLEEP_ENABLE
_HAL_RCC_MIF_CLK_SLEEP_ENABLE
_HAL_RCC_SRAM_CLK_SLEEP_ENABLE
_HAL_RCC_DMA1_CLK_SLEEP_ENABLE
_HAL_RCC_CRC_CLK_SLEEP_DISABLE
_HAL_RCC_MIF_CLK_SLEEP_DISABLE
_HAL_RCC_SRAM_CLK_SLEEP_DISABLE
_HAL_RCC_DMA1_CLK_SLEEP_DISABLE
```

***AHB Peripheral Clock Sleep Enabled or Disabled Status***

```
_HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED
_HAL_RCC_MIF_IS_CLK_SLEEP_ENABLED
_HAL_RCC_SRAM_IS_CLK_SLEEP_ENABLED
_HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED
```

***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 Force Release Reset**

`_HAL_RCC_AHB_FORCE_RESET`  
`_HAL_RCC_DMA1_FORCE_RESET`  
`_HAL_RCC_MIF_FORCE_RESET`  
`_HAL_RCC_CRC_FORCE_RESET`  
`_HAL_RCC_AHB_RELEASE_RESET`  
`_HAL_RCC_CRC_RELEASE_RESET`  
`_HAL_RCC_DMA1_RELEASE_RESET`  
`_HAL_RCC_MIF_RELEASE_RESET`

**AHB Peripheral Clock Enable Disable**

`_HAL_RCC_DMA1_CLK_ENABLE`  
`_HAL_RCC_MIF_CLK_ENABLE`  
`_HAL_RCC_CRC_CLK_ENABLE`  
`_HAL_RCC_DMA1_CLK_DISABLE`  
`_HAL_RCC_MIF_CLK_DISABLE`  
`_HAL_RCC_CRC_CLK_DISABLE`

**AHB Peripheral Clock Enabled or Disabled Status**

`_HAL_RCC_DMA1_IS_CLK_ENABLED`  
`_HAL_RCC_MIF_IS_CLK_ENABLED`  
`_HAL_RCC_CRC_IS_CLK_ENABLED`

**RCC 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 Peripheral Clock Enable Disable**

`_HAL_RCC_WWDG_CLK_ENABLE`  
`_HAL_RCC_PWR_CLK_ENABLE`  
`_HAL_RCC_WWDG_CLK_DISABLE`  
`_HAL_RCC_PWR_CLK_DISABLE`

**APB1 Peripheral Clock Enabled or Disabled Status**

`_HAL_RCC_WWDG_IS_CLK_ENABLED`  
`_HAL_RCC_PWR_IS_CLK_ENABLED`

**APB1 Peripheral Clock Sleep Enable Disable**

\_HAL\_RCC\_WWDG\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_PWR\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_WWDG\_CLK\_SLEEP\_DISABLE  
\_HAL\_RCC\_PWR\_CLK\_SLEEP\_DISABLE

**APB1 Peripheral Clock Sleep Enabled or Disabled Status**

\_HAL\_RCC\_WWDG\_IS\_CLK\_SLEEP\_ENABLED  
\_HAL\_RCC\_PWR\_IS\_CLK\_SLEEP\_ENABLED

**APB1 Peripheral Force Release Reset**

\_HAL\_RCC\_APB1\_FORCE\_RESET  
\_HAL\_RCC\_WWDG\_FORCE\_RESET  
\_HAL\_RCC\_PWR\_FORCE\_RESET  
\_HAL\_RCC\_APB1\_RELEASE\_RESET  
\_HAL\_RCC\_WWDG\_RELEASE\_RESET  
\_HAL\_RCC\_PWR\_RELEASE\_RESET

**APB2 Peripheral Clock Enable Disable**

\_HAL\_RCC\_SYSCFG\_CLK\_ENABLE  
\_HAL\_RCC\_DBGMCU\_CLK\_ENABLE  
\_HAL\_RCC\_SYSCFG\_CLK\_DISABLE  
\_HAL\_RCC\_DBGMCU\_CLK\_DISABLE

**APB2 Peripheral Clock Enabled or Disabled Status**

\_HAL\_RCC\_SYSCFG\_IS\_CLK\_ENABLED  
\_HAL\_RCC\_DBGMCU\_IS\_CLK\_ENABLED

**APB2 Peripheral Clock Sleep Enable Disable**

\_HAL\_RCC\_SYSCFG\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_DBGMCU\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_SYSCFG\_CLK\_SLEEP\_DISABLE  
\_HAL\_RCC\_DBGMCU\_CLK\_SLEEP\_DISABLE

**APB2 Peripheral Clock Sleep Enabled or Disabled Status**

\_HAL\_RCC\_SYSCFG\_IS\_CLK\_SLEEP\_ENABLED  
\_HAL\_RCC\_DBGMCU\_IS\_CLK\_SLEEP\_ENABLED

**APB2 Peripheral Force Release Reset**

\_HAL\_RCC\_APB2\_FORCE\_RESET  
\_HAL\_RCC\_DBGMCU\_FORCE\_RESET  
\_HAL\_RCC\_SYSCFG\_FORCE\_RESET  
\_HAL\_RCC\_APB2\_RELEASE\_RESET  
\_HAL\_RCC\_DBGMCU\_RELEASE\_RESET

---

`_HAL_RCC_SYSCFG_RELEASE_RESET`

**RCC Backup Domain Reset**

`_HAL_RCC_BACKUPRESET_FORCE`

**Notes:**

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC\_CSR register. The BKPSRAM is not affected by this reset.

`_HAL_RCC_BACKUPRESET_RELEASE`

**RCC Exported Macros**

`_HAL_RCC_HSI_ENABLE`

**Notes:**

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

`_HAL_RCC_HSI_DISABLE`

`_HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST`

**Description:**

- Macro to adjust the Internal High Speed oscillator (HSI) calibration value.

**Parameters:**

- `_HSICalibrationValue_`: specifies the calibration trimming value. This parameter must be a number between 0 and 0x1F.

**Notes:**

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

`_HAL_RCC_HSI_CONFIG`

**Description:**

- Macro to enable or disable the Internal High Speed oscillator (HSI).

**Parameters:**

- STATE: specifies the new state of the HSI. This parameter can be one of the following values:
  - RCC\_HSI\_OFF: turn OFF the HSI oscillator
  - RCC\_HSI\_ON: turn ON the HSI oscillator
  - RCC\_HSI\_DIV4: turn ON the HSI oscillator and divide it by 4

**Notes:**

- 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 to clock the PLL and/or system clock. HSI can not be stopped 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 stop the HSI. The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

[\\_\\_HAL\\_RCC\\_MSI\\_ENABLE](#)**Notes:**

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI 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 MSI. After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

[\\_\\_HAL\\_RCC\\_MSI\\_DISABLE](#)[\\_\\_HAL\\_RCC\\_MSI\\_CALIBRATIONVALUE\\_Adjust](#)**Description:**

- Macro Adjusts the Internal Multi Speed oscillator (MSI) calibration value.

**Parameters:**

- MSICalibrationValue: specifies the calibration trimming value. This parameter

---

must be a number between 0 and 0xFF.

**Notes:**

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal MSI RC. Refer to the Application Note AN3300 for more details on how to calibrate the MSI.

[\\_\\_HAL\\_RCC\\_MSI\\_RANGE\\_CONFIG](#)

**Description:**

- Macro to configures the Internal Multi Speed oscillator (MSI) clock range.

**Parameters:**

- [\\_\\_RCC\\_MSIRange](#): specifies the MSI Clock range. This parameter must be one of the following values:
  - [RCC\\_MSIRANGE\\_0](#): MSI clock is around 65.536 KHz
  - [RCC\\_MSIRANGE\\_1](#): MSI clock is around 131.072 KHz
  - [RCC\\_MSIRANGE\\_2](#): MSI clock is around 262.144 KHz
  - [RCC\\_MSIRANGE\\_3](#): MSI clock is around 524.288 KHz
  - [RCC\\_MSIRANGE\\_4](#): MSI clock is around 1.048 MHz
  - [RCC\\_MSIRANGE\\_5](#): MSI clock is around 2.097 MHz (default after Reset or wake-up from STANDBY)
  - [RCC\\_MSIRANGE\\_6](#): MSI clock is around 4.194 MHz

**Notes:**

- After restart from Reset or wakeup from STANDBY, the MSI clock is around 2.097 MHz. The MSI clock does not change after wake-up from STOP mode. The MSI clock range can be modified on the fly.

[\\_\\_HAL\\_RCC\\_GET\\_MSI\\_RANGE](#)

**Description:**

- Macro to get the Internal Multi Speed oscillator ([\\_\\_MSI](#)) clock range in run mode.

**Return value:**

- MSI: clock range. This parameter must be one of the following values:
  - [RCC\\_MSIRANGE\\_0](#): MSI clock is around 65.536 KHz
  - [RCC\\_MSIRANGE\\_1](#): MSI clock is around 131.072 KHz
  - [RCC\\_MSIRANGE\\_2](#): MSI clock is

- around 262.144 KHz
- RCC\_MSIRANGE\_3: MSI clock is around 524.288 KHz
- RCC\_MSIRANGE\_4: MSI clock is around 1.048 MHz
- RCC\_MSIRANGE\_5: MSI clock is around 2.097 MHz (default after Reset or wake-up from STANDBY)
- RCC\_MSIRANGE\_6: MSI clock is around 4.194 MHz

#### \_\_HAL\_RCC\_LSI\_ENABLE

##### **Notes:**

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

#### \_\_HAL\_RCC\_LSI\_DISABLE

#### \_\_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.

#### \_\_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. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL\_PWR\_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC\_LSE\_ON or RCC\_LSE\_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

#### \_\_HAL\_RCC\_RTC\_CLKPRESCALER

##### **Description:**

- Configures or Get the RTC and LCD clock (RTCCLK / LCDCLK).

##### **Parameters:**

- \_\_RTCCLKSOURCE\_\_: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC\_RTCCLKSOURCE\_LSE: LSE selected as RTC clock
  - RCC\_RTCCLKSOURCE\_LSI: LSI selected as RTC clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV2:

- HSE divided by 2 selected as RTC clock
- RCC\_RTCCLKSOURCE\_HSE\_DIV4: HSE divided by 4 selected as RTC clock
- RCC\_RTCCLKSOURCE\_HSE\_DIV8: HSE divided by 8 selected as RTC clock
- RCC\_RTCCLKSOURCE\_HSE\_DIV16: HSE divided by 16 selected as RTC clock

**Notes:**

- As the RTC clock configuration bits are in the RTC domain and write access is denied to this domain after reset, you have to enable write access using PWR\_RTCAccessCmd(ENABLE) function before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the RTC is reset using RCC\_RTCResetCmd function, or by a Power On Reset (POR). The RTC clock (RTCCLK) is used also to clock the LCD (LCDCLK).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

`_HAL_RCC_RTC_CONFIG`  
`_HAL_RCC_GET_RTC_SOURCE`

**Description:**

- Get the RTC and LCD clock (RTCCLK / LCDCLK).

**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\_DIVX : HSE divided by X selected as RTC clock (X can be retrieved thanks to `_HAL_RCC_GET_RTC_HSE_PRE`)

**SCALER()**

[\\_\\_HAL\\_RCC\\_GET\\_RTC\\_HSE\\_PRESCA  
LER](#)

**Description:**

- Get the RTC and LCD HSE clock divider (RTCCLK / LCDCLK).

**Return value:**

- Returned: value can be one of the following values:
  - RCC\_RTC\_HSE\_DIV\_2: HSE divided by 2 selected as RTC clock
  - RCC\_RTC\_HSE\_DIV\_4: HSE divided by 4 selected as RTC clock
  - RCC\_RTC\_HSE\_DIV\_8: HSE divided by 8 selected as RTC clock
  - RCC\_RTC\_HSE\_DIV\_16: HSE divided by 16 selected as RTC clock

[\\_\\_HAL\\_RCC\\_PLL\\_ENABLE](#)

**Notes:**

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

[\\_\\_HAL\\_RCC\\_PLL\\_DISABLE](#)

[\\_\\_HAL\\_RCC\\_PLL\\_CONFIG](#)

**Description:**

- Macro to configure the main PLL clock source, multiplication and division factors.

**Parameters:**

- RCC\_PLLSOURCE: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC\_PLLSOURCE\_HSI: HSI oscillator clock selected as PLL clock entry
  - RCC\_PLLSOURCE\_HSE: HSE oscillator clock selected as PLL clock entry
- PLLMUL: specifies the multiplication factor to generate the PLL VCO clock. This parameter must be one of the following values:
  - RCC\_CFGR\_PLLMUL3: PLLVCO = PLL clock entry x 3
  - RCC\_CFGR\_PLLMUL4: PLLVCO = PLL clock entry x 4
  - RCC\_CFGR\_PLLMUL6: PLLVCO = PLL clock entry x 6

- RCC\_CFGR\_PLLMUL8: PLLVCO = PLL clock entry x 8
- RCC\_CFGR\_PLLMUL12: PLLVCO = PLL clock entry x 12
- RCC\_CFGR\_PLLMUL16: PLLVCO = PLL clock entry x 16
- RCC\_CFGR\_PLLMUL24: PLLVCO = PLL clock entry x 24
- RCC\_CFGR\_PLLMUL32: PLLVCO = PLL clock entry x 32
- RCC\_CFGR\_PLLMUL48: PLLVCO = PLL clock entry x 48
- PLL DIV: specifies the PLL output clock division from PLL VCO clock This parameter must be one of the following values:
  - RCC\_PLLDIV\_2: PLL clock output = PLLVCO / 2
  - RCC\_PLLDIV\_3: PLL clock output = PLLVCO / 3
  - RCC\_PLLDIV\_4: PLL clock output = PLLVCO / 4

**Notes:**

- This function must be used only when the main PLL is disabled.
- The PLL VCO clock frequency must not exceed 96 MHz when the product is in Range 1, 48 MHz when the product is in Range 2 and 24 MHz when the product is in Range 3.

[\\_\\_HAL\\_RCC\\_GET\\_PLL\\_OSCSOURCE](#)**Description:**

- Macro to get the oscillator used as PLL clock source.

**Return value:**

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC\_PLLSOURCE\_HSI: HSI oscillator is used as PLL clock source.
  - RCC\_PLLSOURCE\_HSE: HSE oscillator is used as PLL clock source.

[\\_\\_HAL\\_RCC\\_SYSCLK\\_CONFIG](#)**Description:**

- Macro to configure the system clock source.

**Parameters:**

- SYSCLK SOURCE: specifies the system clock source. This parameter can

be one of the following values:

- RCC\_SYSCLKSOURCE\_MSI: MSI oscillator is used as system clock source.
- 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.

**Return value:**

- None

**\_HAL\_RCC\_GET\_SYSCLK\_SOURCE**

- 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\_MSI: MSI used as system clock.
  - RCC\_SYSCLKSOURCE\_STATUS\_HSI: HSI used as system clock.
  - RCC\_SYSCLKSOURCE\_STATUS\_HSE: HSE used as system clock.
  - RCC\_SYSCLKSOURCE\_STATUS\_PLLCLK: PLL used as system clock.

**\_HAL\_RCC\_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\_CFGR\_MCO\_HSI: HSI clock selected as MCO source
  - RCC\_CFGR\_MCO\_MSI: MSI clock selected as MCO source
  - RCC\_CFGR\_MCO\_HSE: HSE clock selected as MCO source
  - RCC\_CFGR\_MCO\_PLL: PLL clock selected as MCO source
  - RCC\_CFGR\_MCO\_LSI: LSI clock selected as MCO source
  - RCC\_CFGR\_MCO\_LSE: LSE clock selected as MCO source
- \_MCODIV: specifies the MCO clock

prescaler. This parameter can be one of the following values:

- RCC\_CFGR\_MCO\_PRE\_1: no division applied to MCO clock
- RCC\_CFGR\_MCO\_PRE\_2: division by 2 applied to MCO clock
- RCC\_CFGR\_MCO\_PRE\_4: division by 4 applied to MCO clock
- RCC\_CFGR\_MCO\_PRE\_8: division by 8 applied to MCO clock
- RCC\_CFGR\_MCO\_PRE\_16: division by 16 applied to MCO clock

### **RCC Flag**

RCC\_FLAG\_HSIRDY  
 RCC\_FLAG\_HSIDIV  
 RCC\_FLAG\_MSIRDY  
 RCC\_FLAG\_HSERDY  
 RCC\_FLAG\_PLLRDY  
 RCC\_FLAG\_LSERDY  
 RCC\_FLAG\_LSECSS  
 RCC\_FLAG\_LSIRDY  
 RCC\_FLAG\_FWRST  
 RCC\_FLAG\_OBLRST  
 RCC\_FLAG\_PINRST  
 RCC\_FLAG\_PORRST  
 RCC\_FLAG\_SFTRST  
 RCC\_FLAG\_IWDGRST  
 RCC\_FLAG\_WWDGRST  
 RCC\_FLAG\_LPWRRST  
 RCC\_FLAG\_HSI48RDY

### **RCC Flags Interrupts Management**

`_HAL_RCC_ENABLE_IT`

#### **Description:**

- Enable RCC interrupt (Perform Byte access to RCC\_CIER[0:7] bits to enable the selected interrupts).

#### **Parameters:**

- `_INTERRUPT_`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt
  - RCC\_IT\_LSERDY: LSE ready interrupt
  - RCC\_IT\_HSI48RDY: HSI ready interrupt

- RCC\_IT\_HSERDY: HSE ready interrupt
- RCC\_IT\_PLLRDY: PLL ready interrupt
- RCC\_IT\_MSIRDY: MSI ready interrupt
- RCC\_IT\_CSSLSE: LSE CSS interrupt
- RCC\_IT\_HSI48RDY: HSI48 ready interrupt

**Notes:**

- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

**\_HAL\_RCC\_DISABLE\_IT****Description:**

- Disable RCC interrupt (Perform Byte access to RCC\_CIER[0:7] bits to disable the selected interrupts).

**Parameters:**

- \_INTERRUPT\_: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt
  - RCC\_IT\_LSERDY: LSE ready interrupt
  - RCC\_IT\_HSIRDY: HSI ready interrupt
  - RCC\_IT\_HSERDY: HSE ready interrupt
  - RCC\_IT\_PLLRDY: PLL ready interrupt
  - RCC\_IT\_MSIRDY: MSI ready interrupt
  - RCC\_IT\_HSI48RDY: HSI48 ready interrupt
  - RCC\_IT\_CSSLSE: LSE CSS interrupt

**Notes:**

- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

**\_HAL\_RCC\_CLEAR\_IT****Description:**

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC\_CIR[23:16]

bits to clear the selected interrupt pending bits.

#### Parameters:

- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt
  - RCC\_IT\_LSERDY: LSE ready interrupt
  - RCC\_IT\_HSIRDY: HSI ready interrupt
  - RCC\_IT\_HSERDY: HSE ready interrupt
  - RCC\_IT\_PLLRDY: PLL ready interrupt
  - RCC\_IT\_MSIRDY: MSI ready interrupt
  - RCC\_IT\_HSI48RDY: HSI48 ready interrupt
  - RCC\_IT\_CSSLSE: LSE CSS interrupt
  - RCC\_IT\_CSSHSE: 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: PLL ready interrupt
  - RCC\_IT\_MSIRDY: MSI ready interrupt
  - RCC\_IT\_CSSLSE: LSE CSS interrupt
  - RCC\_IT\_CSSHSE: 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\_OBLRST, RCC\_FLAG\_PINRST, RCC\_FLAG\_PORRST, RCC\_FLAG\_SFTRST, RCC\_FLAG\_IWDGRST, RCC\_FLAG\_WWDGRST, RCC\_FLAG\_LPWRST.

### \_\_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\_HSIDIV: HSI clock divider flag
- RCC\_FLAG\_MSIRDY: MSI oscillator clock ready
- RCC\_FLAG\_HSERDY: HSE oscillator clock ready
- RCC\_FLAG\_PLLRDY: PLL clock ready
- RCC\_FLAG\_LSECSS: LSE oscillator clock CSS detected
- RCC\_FLAG\_LSERDY: LSE oscillator clock ready
- RCC\_FLAG\_LSIRDY: LSI oscillator clock ready
- RCC\_FLAG\_FWRST: Firewall reset
- RCC\_FLAG\_OBLRST: Option Byte Loader (OBL) reset
- RCC\_FLAG\_PINRST: Pin reset
- RCC\_FLAG\_PORRST: POR/PDR reset
- RCC\_FLAG\_SFTRST: Software reset
- RCC\_FLAG\_IWDGRST: Independent Watchdog reset
- RCC\_FLAG\_WWDGRST: Window Watchdog reset
- RCC\_FLAG\_LPWRRST: Low Power reset

#### **Return value:**

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

#### **RCC HSE Config**

RCC\_HSE\_OFF

RCC\_HSE\_ON

RCC\_HSE\_BYPASS

#### **RCC HSI48 Configuration**

RCC\_HSI48\_OFF

RCC\_HSI48\_ON

#### **RCC HSI Configuration**

RCC\_HSI\_OFF

RCC\_HSI\_ON

RCC\_HSI\_DIV4

RCC\_HSI\_OUTEN

#### **RCC Interruptions**

RCC\_IT\_LSIRDY  
RCC\_IT\_LSERDY  
RCC\_IT\_HSIRDY  
RCC\_IT\_HSERDY  
RCC\_IT\_PLLRDY  
RCC\_IT\_MSIRDY  
RCC\_IT\_CSSLSE  
RCC\_IT\_CSSHSE  
RCC\_IT\_HSI48RDY

**IOPORT Peripheral Clock Enable Disable**

\_HAL\_RCC\_GPIOA\_CLK\_ENABLE  
\_HAL\_RCC\_GPIOB\_CLK\_ENABLE  
\_HAL\_RCC\_GPIOC\_CLK\_ENABLE  
\_HAL\_RCC\_GPIOH\_CLK\_ENABLE  
\_HAL\_RCC\_GPIOA\_CLK\_DISABLE  
\_HAL\_RCC\_GPIOB\_CLK\_DISABLE  
\_HAL\_RCC\_GPIOC\_CLK\_DISABLE  
\_HAL\_RCC\_GPIOH\_CLK\_DISABLE

**IOPORT Peripheral Clock Sleep Enable Disable**

\_HAL\_RCC\_GPIOA\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_GPIOB\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_GPIOC\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_GPIOH\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_GPIOA\_CLK\_SLEEP\_DISABLE  
\_HAL\_RCC\_GPIOB\_CLK\_SLEEP\_DISABLE  
\_HAL\_RCC\_GPIOC\_CLK\_SLEEP\_DISABLE  
\_HAL\_RCC\_GPIOH\_CLK\_SLEEP\_DISABLE

**IOPORT Peripheral Clock Sleep Enabled or Disabled Status**

\_HAL\_RCC\_GPIOA\_IS\_CLK\_SLEEP\_ENABLED  
\_HAL\_RCC\_GPIOB\_IS\_CLK\_SLEEP\_ENABLED  
\_HAL\_RCC\_GPIOC\_IS\_CLK\_SLEEP\_ENABLED  
\_HAL\_RCC\_GPIOH\_IS\_CLK\_SLEEP\_ENABLED

**IOPORT Peripheral Force Release Reset**

\_HAL\_RCC\_IOP\_FORCE\_RESET  
\_HAL\_RCC\_GPIOA\_FORCE\_RESET  
\_HAL\_RCC\_GPIOB\_FORCE\_RESET

\_HAL\_RCC\_GPIOC\_FORCE\_RESET  
\_HAL\_RCC\_GPIOH\_FORCE\_RESET  
\_HAL\_RCC\_IOP\_RELEASE\_RESET  
\_HAL\_RCC\_GPIOA\_RELEASE\_RESET  
\_HAL\_RCC\_GPIOB\_RELEASE\_RESET  
\_HAL\_RCC\_GPIOC\_RELEASE\_RESET  
\_HAL\_RCC\_GPIOH\_RELEASE\_RESET

**IOPORT Peripheral Clock Enabled or Disabled Status**

\_HAL\_RCC\_GPIOA\_IS\_CLK\_ENABLED  
\_HAL\_RCC\_GPIOB\_IS\_CLK\_ENABLED  
\_HAL\_RCC\_GPIOC\_IS\_CLK\_ENABLED  
\_HAL\_RCC\_GPIOH\_IS\_CLK\_ENABLED

**RCC LSE Config**

RCC\_LSE\_OFF  
RCC\_LSE\_ON  
RCC\_LSE\_BYPASS

**RCC LSI Config**

RCC\_LSI\_OFF  
RCC\_LSI\_ON  
RCC\_MSICALIBRATION\_DEFAULT

**RCC MCO1 Clock Source**

RCC\_MCO1SOURCE\_NOCLOCK  
RCC\_MCO1SOURCE\_SYSCLK  
RCC\_MCO1SOURCE\_HSI  
RCC\_MCO1SOURCE\_MSI  
RCC\_MCO1SOURCE\_HSE  
RCC\_MCO1SOURCE\_PLLCLK  
RCC\_MCO1SOURCE\_LSI  
RCC\_MCO1SOURCE\_LSE  
RCC\_MCO1SOURCE\_HSI48

**RCC MCO Prescaler**

RCC\_MCODIV\_1  
RCC\_MCODIV\_2  
RCC\_MCODIV\_4  
RCC\_MCODIV\_8  
RCC\_MCODIV\_16

**RCC MCO Index**

RCC\_MCO1

RCC\_MCO2

RCC\_MCO3

***RCC MSI Clock Range***

RCC\_MSIRANGE\_0    MSI = 65.536 KHz

RCC\_MSIRANGE\_1    MSI = 131.072 KHz

RCC\_MSIRANGE\_2    MSI = 262.144 KHz

RCC\_MSIRANGE\_3    MSI = 524.288 KHz

RCC\_MSIRANGE\_4    MSI = 1.048 MHz

RCC\_MSIRANGE\_5    MSI = 2.097 MHz

RCC\_MSIRANGE\_6    MSI = 4.194 MHz

***RCC MSI Config***

RCC\_MSI\_OFF

RCC\_MSI\_ON

RCC\_HSICALIBRATION\_DEFAULT

***Oscillator Type***

RCC\_OSCILLATORTYPE\_NONE    Oscillator configuration unchanged

RCC\_OSCILLATORTYPE\_HSE    HSE to configure

RCC\_OSCILLATORTYPE\_HSI    HSI to configure

RCC\_OSCILLATORTYPE\_LSE    LSE to configure

RCC\_OSCILLATORTYPE\_LSI    LSI to configure

RCC\_OSCILLATORTYPE\_MSI    MSI to configure

RCC\_OSCILLATORTYPE\_HSI48

***RCC PLL Dividers***

RCC\_PLLDIV\_2

RCC\_PLLDIV\_3

RCC\_PLLDIV\_4

***RCC PLL Multipliers***

RCC\_PLLMUL\_3

RCC\_PLLMUL\_4

RCC\_PLLMUL\_6

RCC\_PLLMUL\_8

RCC\_PLLMUL\_12

RCC\_PLLMUL\_16

RCC\_PLLMUL\_24

RCC\_PLLMUL\_32



---

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure
RCC_CLOCKTYPE_PCLK2	PCLK2 to configure

***Timeout Values***

RCC_DBP_TIMEOUT_VALUE
RCC_LSE_TIMEOUT_VALUE
RCC_HSE_TIMEOUT_VALUE

## 37 HAL RCC Extension Driver

### 37.1 RCCEEx Firmware driver registers structures

#### 37.1.1 RCC\_PeriphCLKInitTypeDef

##### Data Fields

- *uint32\_t PeriphClockSelection*
- *uint32\_t Usart1ClockSelection*
- *uint32\_t Usart2ClockSelection*
- *uint32\_t Lpuart1ClockSelection*
- *uint32\_t I2c1ClockSelection*
- *uint32\_t I2c3ClockSelection*
- *uint32\_t RTCClockSelection*
- *uint32\_t LCDClockSelection*
- *uint32\_t UsbClockSelection*
- *uint32\_t LptimClockSelection*

##### Field Documentation

- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PeriphClockSelection***  
The Extended Clock to be configured. This parameter can be a value of [\*\*RCCEEx\\_Periph\\_Clock\\_Selection\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Usart1ClockSelection***  
USART1 clock source This parameter can be a value of [\*\*RCCEEx\\_USART1\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Usart2ClockSelection***  
USART2 clock source This parameter can be a value of [\*\*RCCEEx\\_USART2\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Lpuart1ClockSelection***  
LPUART1 clock source This parameter can be a value of [\*\*RCCEEx\\_LPUART1\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::I2c1ClockSelection***  
I2C1 clock source This parameter can be a value of [\*\*RCCEEx\\_I2C1\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::I2c3ClockSelection***  
I2C3 clock source This parameter can be a value of [\*\*RCCEEx\\_I2C3\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::RTCClockSelection***  
Specifies RTC Clock Prescalers Selection This parameter can be a value of [\*\*RCC\\_RTC\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::LCDClockSelection***  
specifies the LCD clock source. This parameter can be a value of [\*\*RCC\\_RTC\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::UsbClockSelection***  
Specifies USB and RNG Clock Selection This parameter can be a value of [\*\*RCCEEx\\_USB\\_Clock\\_Source\*\*](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::LptimClockSelection***  
LPTIM1 clock source This parameter can be a value of [\*\*RCCEEx\\_LPTIM1\\_Clock\\_Source\*\*](#)

### 37.1.2 RCC\_CRSInitTypeDef

#### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Source*
- *uint32\_t Polarity*
- *uint32\_t ReloadValue*
- *uint32\_t ErrorLimitValue*
- *uint32\_t HSI48CalibrationValue*

#### Field Documentation

- ***uint32\_t RCC\_CRSInitTypeDef::Prescaler***  
Specifies the division factor of the SYNC signal. This parameter can be a value of [\*RCCEEx\\_CRS\\_SynchroDivider\*](#)
- ***uint32\_t RCC\_CRSInitTypeDef::Source***  
Specifies the SYNC signal source. This parameter can be a value of [\*RCCEEx\\_CRS\\_SynchroSource\*](#)
- ***uint32\_t RCC\_CRSInitTypeDef::Polarity***  
Specifies the input polarity for the SYNC signal source. This parameter can be a value of [\*RCCEEx\\_CRS\\_SynchroPolarity\*](#)
- ***uint32\_t RCC\_CRSInitTypeDef::ReloadValue***  
Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro  
`_HAL_RCC_CRS_CALCULATE_RELOADVALUE(_FTARGET_, _FSYNC_)` This parameter must be a number between 0 and 0xFFFF or a value of [\*RCCEEx\\_CRS\\_ReloadValueDefault\*](#).
- ***uint32\_t RCC\_CRSInitTypeDef::ErrorLimitValue***  
Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of [\*RCCEEx\\_CRS\\_ErrorLimitDefault\*](#)
- ***uint32\_t RCC\_CRSInitTypeDef::HSI48CalibrationValue***  
Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of [\*RCCEEx\\_CRS\\_HSI48CalibrationDefault\*](#)

### 37.1.3 RCC\_CRSSynchroInfoTypeDef

#### Data Fields

- *uint32\_t ReloadValue*
- *uint32\_t HSI48CalibrationValue*
- *uint32\_t FreqErrorCapture*
- *uint32\_t FreqErrorDirection*

#### Field Documentation

- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::ReloadValue***  
Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF

- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::HSI48CalibrationValue***  
Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::FreqErrorCapture***  
Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::FreqErrorDirection***  
Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of [RCCEEx\\_CRS\\_FreqErrorDirection](#)

## 37.2 RCCEEx Firmware driver API description

### 37.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- [HAL\\_RCCEEx\\_PерiphCLKConfig\(\)](#)
- [HAL\\_RCCEEx\\_GetPeriphCLKConfig\(\)](#)
- [HAL\\_RCCEEx\\_GetPeriphCLKFreq\(\)](#)
- [HAL\\_RCCEEx\\_EnableLSECSS\(\)](#)
- [HAL\\_RCCEEx\\_DisableLSECSS\(\)](#)
- [HAL\\_RCCEEx\\_EnableLSECSS\\_IT\(\)](#)
- [HAL\\_RCCEEx\\_LSECSS\\_IRQHandler\(\)](#)
- [HAL\\_RCCEEx\\_LSECSS\\_Callback\(\)](#)
- [HAL\\_RCCEEx\\_CRSConfig\(\)](#)
- [HAL\\_RCCEEx\\_CRSSoftwareSynchronizationGenerate\(\)](#)
- [HAL\\_RCCEEx\\_CRSGetSynchronizationInfo\(\)](#)
- [HAL\\_RCCEEx\\_CRSWaitSynchronization\(\)](#)
- [HAL\\_RCCEEx\\_EnableHSI48\\_VREFINT\(\)](#)
- [HAL\\_RCCEEx\\_DisableHSI48\\_VREFINT\(\)](#)

### 37.2.2 Detailed description of functions

#### HAL\_RCCEEx\_PерiphCLKConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCCEEx_PерiphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code>
Function Description	Initializes the RCC extended peripherals clocks.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit:</b> pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, I2C3, RTC, USB/RNG and LPTIM1 clocks).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.</li> <li>• If HAL_ERROR returned, first switch-OFF HSE clock</li> </ul>

oscillator with HAL\_RCC\_OscConfig() to possibly update HSE divider.

### **HAL\_RCCEEx\_GetPeriphCLKConfig**

Function Name	<b>void HAL_RCCEEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</b>
Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit:</b> pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, I2C3, RTC, USB/RNG and LPTIM1 clocks).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_RCCEEx\_GetPeriphCLKFreq**

Function Name	<b>uint32_t HAL_RCCEEx_GetPeriphCLKFreq (uint32_t PeriphClk)</b>
Function Description	Return the peripheral clock frequency for some peripherals.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClk:</b> Peripheral clock identifier This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- RCC_PERIPHCLK_RTC: RTC peripheral clock</li> <li>- RCC_PERIPHCLK_LCD: LCD peripheral clock (*)</li> <li>- RCC_PERIPHCLK_USB: USB or RNG peripheral clock (*)</li> <li>- RCC_PERIPHCLK_USART1: USART1 peripheral clock (*)</li> <li>- RCC_PERIPHCLK_USART2: USART2 peripheral clock</li> <li>- RCC_PERIPHCLK_LPUART1: LPUART1 peripheral clock</li> <li>- RCC_PERIPHCLK_I2C1: I2C1 peripheral clock</li> <li>- RCC_PERIPHCLK_I2C2: I2C2 peripheral clock (*)</li> <li>- RCC_PERIPHCLK_I2C3: I2C3 peripheral clock (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Frequency:</b> in Hz (0: means that no available frequency for the peripheral)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Return 0 if peripheral clock identifier not managed by this API</li> <li>• (*) means that this peripheral is not present on all the STM32L0xx devices</li> </ul>

### **HAL\_RCCEEx\_EnableLSECSS**

Function Name	<b>void HAL_RCCEEx_EnableLSECSS (void )</b>
Function Description	Enables the LSE Clock Security System.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_RCCEEx\_DisableLSECSS**

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

---

Function Description	Disables the LSE Clock Security System.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_RCCEEx\_EnableLSECSS\_IT**

Function Name	<b>void HAL_RCCEEx_EnableLSECSS_IT (void )</b>
Function Description	Enable the LSE Clock Security System IT & corresponding EXTI line.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• LSE Clock Security System IT is mapped on RTC EXTI line 19</li> </ul>

### **HAL\_RCCEEx\_LSECSS\_IRQHandler**

Function Name	<b>void HAL_RCCEEx_LSECSS_IRQHandler (void )</b>
Function Description	Handle the RCC LSE Clock Security System interrupt request.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_RCCEEx\_LSECSS\_Callback**

Function Name	<b>void HAL_RCCEEx_LSECSS_Callback (void )</b>
Function Description	RCCEx LSE Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> <li>• <b>none:</b></li> </ul>

### **HAL\_RCCEEx\_CRSConfig**

Function Name	<b>void HAL_RCCEEx_CRSConfig (RCC_CRSInitTypeDef * pInit)</b>
Function Description	Start automatic synchronization using polling mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>pInit:</b> Pointer on RCC_CRSInitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_RCCEEx\_CRSSoftwareSynchronizationGenerate**

Function Name	<b>void HAL_RCCEEx_CRSSoftwareSynchronizationGenerate (void )</b>
Function Description	Generate the software synchronization event.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_RCCEEx\_CRSGetSynchronizationInfo**

Function Name	<b>void HAL_RCCEEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)</b>
Function Description	Function to return synchronization info.
Parameters	<ul style="list-style-type: none"> <li>• <b>pSynchroInfo:</b> Pointer on RCC_CRSSynchroInfoTypeDef structure</li> </ul>

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

### **HAL\_RCCEEx\_CRSWaitSynchronization**

Function Name	<b>uint32_t HAL_RCCEEx_CRSWaitSynchronization (uint32_t Timeout)</b>
Function Description	This function handles CRS Synchronization Timeout.
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout:</b> Duration of the timeout</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Combination:</b> of Synchronization status This parameter can be a combination of the following values:           <ul style="list-style-type: none"> <li>- RCC_CRS_TIMEOUT</li> <li>- RCC_CRS_SYNCOK</li> <li>- RCC_CRS_SYNCWARN</li> <li>- RCC_CRS_SYNCERR</li> <li>- RCC_CRS_SYNCMISS</li> <li>- RCC_CRS_TRIMOVF</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.</li> <li>• If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.</li> </ul>

### **HAL\_RCCEEx\_EnableHSI48\_VREFINT**

Function Name	<b>void HAL_RCCEEx_EnableHSI48_VREFINT (void )</b>
Function Description	Enables Vrefint for the HSI48.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This is functional only if the LOCK is not set</li> </ul>

### **HAL\_RCCEEx\_DisableHSI48\_VREFINT**

Function Name	<b>void HAL_RCCEEx_DisableHSI48_VREFINT (void )</b>
Function Description	Disables the Vrefint for the HSI48.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This is functional only if the LOCK is not set</li> </ul>

## **37.3 RCCEx Firmware driver defines**

### **37.3.1 RCCEx**

#### **AHB Peripheral Clock Sleep Enable Disable**

```
_HAL_RCC_TSC_CLK_SLEEP_ENABLE
_HAL_RCC_RNG_CLK_SLEEP_ENABLE
_HAL_RCC_TSC_CLK_SLEEP_DISABLE
_HAL_RCC_RNG_CLK_SLEEP_DISABLE
_HAL_RCC_AES_CLK_SLEEP_ENABLE
```

\_HAL\_RCC\_AES\_CLK\_SLEEP\_DISABLE  
**AHB Peripheral Force Release Reset**  
\_HAL\_RCC\_AES\_FORCE\_RESET  
\_HAL\_RCC\_AES\_RELEASE\_RESET  
\_HAL\_RCC\_TSC\_FORCE\_RESET  
\_HAL\_RCC\_TSC\_RELEASE\_RESET  
\_HAL\_RCC\_RNG\_FORCE\_RESET  
\_HAL\_RCC\_RNG\_RELEASE\_RESET  
**APB1 Peripheral Clock Enable Disable**  
\_HAL\_RCC\_USB\_CLK\_ENABLE  
\_HAL\_RCC\_USB\_CLK\_DISABLE  
\_HAL\_RCC\_CRS\_CLK\_ENABLE  
\_HAL\_RCC\_CRS\_CLK\_DISABLE  
\_HAL\_RCC\_LCD\_CLK\_ENABLE  
\_HAL\_RCC\_LCD\_CLK\_DISABLE  
\_HAL\_RCC\_TIM2\_CLK\_ENABLE  
\_HAL\_RCC\_TIM3\_CLK\_ENABLE  
\_HAL\_RCC\_TIM6\_CLK\_ENABLE  
\_HAL\_RCC\_TIM7\_CLK\_ENABLE  
\_HAL\_RCC\_SPI2\_CLK\_ENABLE  
\_HAL\_RCC\_USART2\_CLK\_ENABLE  
\_HAL\_RCC\_USART4\_CLK\_ENABLE  
\_HAL\_RCC\_USART5\_CLK\_ENABLE  
\_HAL\_RCC\_LPUART1\_CLK\_ENABLE  
\_HAL\_RCC\_I2C1\_CLK\_ENABLE  
\_HAL\_RCC\_I2C2\_CLK\_ENABLE  
\_HAL\_RCC\_I2C3\_CLK\_ENABLE  
\_HAL\_RCC\_DAC\_CLK\_ENABLE  
\_HAL\_RCC\_LPTIM1\_CLK\_ENABLE  
\_HAL\_RCC\_TIM2\_CLK\_DISABLE  
\_HAL\_RCC\_TIM3\_CLK\_DISABLE  
\_HAL\_RCC\_TIM6\_CLK\_DISABLE  
\_HAL\_RCC\_TIM7\_CLK\_DISABLE  
\_HAL\_RCC\_SPI2\_CLK\_DISABLE  
\_HAL\_RCC\_USART2\_CLK\_DISABLE  
\_HAL\_RCC\_USART4\_CLK\_DISABLE

```
_HAL_RCC_USART5_CLK_DISABLE  
_HAL_RCC_LPUART1_CLK_DISABLE  
_HAL_RCC_I2C1_CLK_DISABLE  
_HAL_RCC_I2C2_CLK_DISABLE  
_HAL_RCC_I2C3_CLK_DISABLE  
_HAL_RCC_DAC_CLK_DISABLE  
_HAL_RCC_LPTIM1_CLK_DISABLE  
APB1 Peripheral Clock Sleep Enable Disable  
_HAL_RCC_TIM2_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM3_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM6_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM7_CLK_SLEEP_ENABLE  
_HAL_RCC_SPI2_CLK_SLEEP_ENABLE  
_HAL_RCC_USART2_CLK_SLEEP_ENABLE  
_HAL_RCC_USART4_CLK_SLEEP_ENABLE  
_HAL_RCC_USART5_CLK_SLEEP_ENABLE  
_HAL_RCC_LPUART1_CLK_SLEEP_ENABLE  
_HAL_RCC_I2C1_CLK_SLEEP_ENABLE  
_HAL_RCC_I2C2_CLK_SLEEP_ENABLE  
_HAL_RCC_I2C3_CLK_SLEEP_ENABLE  
_HAL_RCC_DAC_CLK_SLEEP_ENABLE  
_HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM2_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM3_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM6_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM7_CLK_SLEEP_DISABLE  
_HAL_RCC_SPI2_CLK_SLEEP_DISABLE  
_HAL_RCC_USART2_CLK_SLEEP_DISABLE  
_HAL_RCC_USART4_CLK_SLEEP_DISABLE  
_HAL_RCC_USART5_CLK_SLEEP_DISABLE  
_HAL_RCC_LPUART1_CLK_SLEEP_DISABLE  
_HAL_RCC_I2C1_CLK_SLEEP_DISABLE  
_HAL_RCC_I2C2_CLK_SLEEP_DISABLE  
_HAL_RCC_I2C3_CLK_SLEEP_DISABLE  
_HAL_RCC_DAC_CLK_SLEEP_DISABLE  
_HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE
```

```
_HAL_RCC_USB_CLK_SLEEP_ENABLE  
_HAL_RCC_USB_CLK_SLEEP_DISABLE  
_HAL_RCC_CRS_CLK_SLEEP_ENABLE  
_HAL_RCC_CRS_CLK_SLEEP_DISABLE  
_HAL_RCC_LCD_CLK_SLEEP_ENABLE  
_HAL_RCC_LCD_CLK_SLEEP_DISABLE  
APB1 Peripheral Force Release Reset  
_HAL_RCC_TIM2_FORCE_RESET  
_HAL_RCC_TIM3_FORCE_RESET  
_HAL_RCC_TIM6_FORCE_RESET  
_HAL_RCC_TIM7_FORCE_RESET  
_HAL_RCC_LPTIM1_FORCE_RESET  
_HAL_RCC_I2C1_FORCE_RESET  
_HAL_RCC_I2C2_FORCE_RESET  
_HAL_RCC_I2C3_FORCE_RESET  
_HAL_RCC_USART2_FORCE_RESET  
_HAL_RCC_USART4_FORCE_RESET  
_HAL_RCC_USART5_FORCE_RESET  
_HAL_RCC_LPUART1_FORCE_RESET  
_HAL_RCC_SPI2_FORCE_RESET  
_HAL_RCC_DAC_FORCE_RESET  
_HAL_RCC_TIM2_RELEASE_RESET  
_HAL_RCC_TIM3_RELEASE_RESET  
_HAL_RCC_TIM6_RELEASE_RESET  
_HAL_RCC_TIM7_RELEASE_RESET  
_HAL_RCC_LPTIM1_RELEASE_RESET  
_HAL_RCC_I2C1_RELEASE_RESET  
_HAL_RCC_I2C2_RELEASE_RESET  
_HAL_RCC_I2C3_RELEASE_RESET  
_HAL_RCC_USART2_RELEASE_RESET  
_HAL_RCC_USART4_RELEASE_RESET  
_HAL_RCC_USART5_RELEASE_RESET  
_HAL_RCC_LPUART1_RELEASE_RESET  
_HAL_RCC_SPI2_RELEASE_RESET  
_HAL_RCC_DAC_RELEASE_RESET  
_HAL_RCC_USB_FORCE_RESET
```

\_\_HAL\_RCC\_USB\_RELEASE\_RESET  
\_\_HAL\_RCC\_CRS\_FORCE\_RESET  
\_\_HAL\_RCC\_CRS\_RELEASE\_RESET  
\_\_HAL\_RCC\_LCD\_FORCE\_RESET  
\_\_HAL\_RCC\_LCD\_RELEASE\_RESET

**APB2 Peripheral Clock Enable Disable**

\_\_HAL\_RCC\_TIM21\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM22\_CLK\_ENABLE  
\_\_HAL\_RCC\_ADC1\_CLK\_ENABLE  
\_\_HAL\_RCC\_SPI1\_CLK\_ENABLE  
\_\_HAL\_RCC\_USART1\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM21\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM22\_CLK\_DISABLE  
\_\_HAL\_RCC\_ADC1\_CLK\_DISABLE  
\_\_HAL\_RCC\_SPI1\_CLK\_DISABLE  
\_\_HAL\_RCC\_USART1\_CLK\_DISABLE  
\_\_HAL\_RCC\_FIREWALL\_CLK\_ENABLE  
\_\_HAL\_RCC\_FIREWALL\_CLK\_DISABLE

**APB2 Peripheral Clock Sleep Enable Disable**

\_\_HAL\_RCC\_TIM21\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM22\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_ADC1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_SPI1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_USART1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM21\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM22\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_ADC1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SPI1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_USART1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_LCD\_CONFIG

**Description:**

- Macro to configures LCD clock (LCDCLK).

**Parameters:**

- LCD\_CLKSOURCE: specifies the LCD clock source. This parameter can be one of the following values:
  - RCC\_RTCCLKSOURCE\_LSE LSE selected as LCD clock
  - RCC\_RTCCLKSOURCE\_LSI LSI selected as LCD clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV 2 HSE divided by 2 selected as LCD clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV 4 HSE divided by 4 selected as LCD clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV 8 HSE divided by 8 selected as LCD clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV 16 HSE divided by 16 selected as LCD clock

**Notes:**

- LCD and RTC use the same configuration LCD can however be used in the Stop low power mode if the LSE or LSI is used as the LCD clock source.

\_HAL\_RCC\_GET\_LCD\_SOURCE  
\_HAL\_RCC\_GET\_LCD\_HSE\_PRESCALE

**APB2 Peripheral Force Release Reset**

\_HAL\_RCC\_USART1\_FORCE\_RESET  
\_HAL\_RCC\_ADC1\_FORCE\_RESET  
\_HAL\_RCC\_SPI1\_FORCE\_RESET  
\_HAL\_RCC\_TIM21\_FORCE\_RESET  
\_HAL\_RCC\_TIM22\_FORCE\_RESET  
\_HAL\_RCC\_USART1\_RELEASE\_RESET  
\_HAL\_RCC\_ADC1\_RELEASE\_RESET  
\_HAL\_RCC\_SPI1\_RELEASE\_RESET  
\_HAL\_RCC\_TIM21\_RELEASE\_RESET  
\_HAL\_RCC\_TIM22\_RELEASE\_RESET

**RCC CRS Error Limit Default**

RCC\_CRS\_ERRORLIMIT\_DEFAULT Default Frequency error limit

**RCC CRS Flags**

---

RCC_CRS_FLAG_SYNCOK	
RCC_CRS_FLAG_SYNCWARN	
RCC_CRS_FLAG_ERR	
RCC_CRS_FLAG_ESYNC	
RCC_CRS_FLAG_TRIMOVF	Trimming overflow or underflow
RCC_CRS_FLAG_SYNCERR	SYNC error
RCC_CRS_FLAG_SYNCMISS	SYNC missed

***RCC CRS Frequency Error Direction***

RCC_CRS_FREQERRORDIR_UP	Upcounting direction, the actual frequency is above the target
RCC_CRS_FREQERRORDIR_DOWN	Downcounting direction, the actual frequency is below the target

***RCC CRS HSI48 Calibration Default***

RCC_CRS_HSI48CALIBRATION_DEFAULT	The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency
----------------------------------	--

***RCC CRS Interrupt Sources***

RCC_CRS_IT_SYNCOK	SYNC event OK
RCC_CRS_IT_SYNCWARN	SYNC warning
RCC_CRS_IT_ERR	error
RCC_CRS_IT_ESYNC	Expected SYNC
RCC_CRS_IT_TRIMOVF	Trimming overflow or underflow
RCC_CRS_IT_SYNCERR	SYNC error
RCC_CRS_IT_SYNCMISS	SYNC missed

***RCC CRS Reload Default Value***

RCC_CRS_RELOADVALUE_DEFAULT	The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).
-----------------------------	--

***RCC CRS Synchro Divider***

RCC_CRS_SYNC_DIV1	Synchro Signal not divided (default)
RCC_CRS_SYNC_DIV2	Synchro Signal divided by 2
RCC_CRS_SYNC_DIV4	Synchro Signal divided by 4
RCC_CRS_SYNC_DIV8	Synchro Signal divided by 8
RCC_CRS_SYNC_DIV16	Synchro Signal divided by 16
RCC_CRS_SYNC_DIV32	Synchro Signal divided by 32
RCC_CRS_SYNC_DIV64	Synchro Signal divided by 64

RCC\_CRS\_SYNC\_DIV128 Synchro Signal divided by 128

#### **RCC CRS Synchro Polarity**

RCC\_CRS\_SYNC\_POLARITY\_RISING Synchro Active on rising edge (default)

RCC\_CRS\_SYNC\_POLARITY\_FALLING Synchro Active on falling edge

#### **RCC CRS Synchro Source**

RCC\_CRS\_SYNC\_SOURCE\_GPIO Synchro Signal source GPIO

RCC\_CRS\_SYNC\_SOURCE\_LSE Synchro Signal source LSE

RCC\_CRS\_SYNC\_SOURCE\_USB Synchro Signal source USB SOF (default)

#### **RCCEx Exported Constants**

RCC\_CRS\_NONE

RCC\_CRS\_TIMEOUT

RCC\_CRS\_SYNCOK

RCC\_CRS\_SYNCWARN

RCC\_CRS\_SYNCERR

RCC\_CRS\_SYNCMISS

RCC\_CRS\_TRIMOVF

#### **RCCEx Exported Macros**

\_HAL\_RCC\_I2C1\_CONFIG

##### **Description:**

- Macro to configure the I2C1 clock (I2C1CLK).

##### **Parameters:**

- \_I2C1\_CLKSOURCE\_: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC\_I2C1CLKSOURCE\_PCLK  
1: PCLK1 selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_HSI:  
HSI selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_SYSC  
LK: System Clock selected as I2C1 clock

##### **Return value:**

- None

\_HAL\_RCC\_GET\_I2C1\_SOURCE

##### **Description:**

- Macro to get the I2C1 clock source.

##### **Return value:**

- The: clock source can be one of the following values:
  - RCC\_I2C1CLKSOURCE\_PCLK  
1: PCLK1 selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_HSI:

- HSI selected as I2C1 clock
- RCC\_I2C1CLKSOURCE\_SYSC
- LK: System Clock selected as I2C1 clock

### \_HAL\_RCC\_I2C3\_CONFIG

**Description:**

- Macro to configure the I2C3 clock (I2C3CLK).

**Parameters:**

- \_I2C3\_CLKSOURCE\_: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC\_I2C3CLKSOURCE\_PCLK 1: PCLK1 selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_HSI: HSI selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_SYSC LK: System Clock selected as I2C3 clock

**Return value:**

- None

### \_HAL\_RCC\_GET\_I2C3\_SOURCE

**Description:**

- Macro to get the I2C3 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_I2C3CLKSOURCE\_PCLK 1: PCLK1 selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_HSI: HSI selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_SYSC LK: System Clock selected as I2C3 clock

### \_HAL\_RCC\_USART1\_CONFIG

**Description:**

- Macro to configure the USART1 clock (USART1CLK).

**Parameters:**

- \_USART1\_CLKSOURCE\_: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC\_USART1CLKSOURCE\_P CLK2: PCLK2 selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_H SI: HSI selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_S

- YSCLK: System Clock selected as USART1 clock
- RCC\_USART1CLKSOURCE\_LS  
E: LSE selected as USART1 clock

**Return value:**

- None

[\\_\\_HAL\\_RCC\\_GET\\_USART1\\_SOURCE](#)**Description:**

- Macro to get the USART1 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USART1CLKSOURCE\_P  
CLK2: PCLK2 selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_H  
SI: HSI selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_S  
YSCLK: System Clock selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_LS  
E: LSE selected as USART1 clock

[\\_\\_HAL\\_RCC\\_USART2\\_CONFIG](#)**Description:**

- Macro to configure the USART2 clock (USART2CLK).

**Parameters:**

- [\\_\\_USART2\\_CLKSOURCE\\_\\_](#): specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC\_USART2CLKSOURCE\_P  
CLK1: PCLK1 selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_H  
SI: HSI selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_S  
YSCLK: System Clock selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_LS  
E: LSE selected as USART2 clock

**Return value:**

- None

\_\_HAL\_RCC\_GET\_USART2\_SOURCE**Description:**

- Macro to get the USART2 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USART2CLKSOURCE\_P CLK1: PCLK1 selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_H SI: HSI selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_S YSCLK: System Clock selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_LS E: LSE selected as USART2 clock

\_\_HAL\_RCC\_LPUART1\_CONFIG**Description:**

- Macro to configure the LPUART1 clock (LPUART1CLK).

**Parameters:**

- \_\_LPUART1\_CLKSOURCE\_\_: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC\_LPUART1CLKSOURCE\_P CLK1: PCLK1 selected as LPUART1 clock
  - RCC\_LPUART1CLKSOURCE\_H SI: HSI selected as LPUART1 clock
  - RCC\_LPUART1CLKSOURCE\_S YSCLK: System Clock selected as LPUART1 clock
  - RCC\_LPUART1CLKSOURCE\_L SE: LSE selected as LPUART1 clock

**Return value:**

- None

\_\_HAL\_RCC\_GET\_LPUART1\_SOURCE**Description:**

- Macro to get the LPUART1 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_LPUART1CLKSOURCE\_P CLK1: PCLK1 selected as

LPUART1 clock

- RCC\_LPUART1CLKSOURCE\_H  
SI: HSI selected as LPUART1 clock
- RCC\_LPUART1CLKSOURCE\_S  
YSCLK: System Clock selected as LPUART1 clock
- RCC\_LPUART1CLKSOURCE\_L  
SE: LSE selected as LPUART1 clock

#### \_HAL\_RCC\_LPTIM1\_CONFIG

##### Description:

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

##### Parameters:

- \_LPTIM1\_CLKSOURCE\_: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC\_LPTIM1CLKSOURCE\_PC  
LK: PCLK selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSI : HSI selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_HSI : LSI selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSE : LSE selected as LPTIM1 clock

##### Return value:

- None

#### \_HAL\_RCC\_GET\_LPTIM1\_SOURCE

##### Description:

- Macro to get the LPTIM1 clock source.

##### Return value:

- The: clock source can be one of the following values:
  - RCC\_LPTIM1CLKSOURCE\_PC  
LK: PCLK selected as LPUART1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSI : HSI selected as LPUART1 clock
  - RCC\_LPTIM1CLKSOURCE\_HSI : System Clock selected as LPUART1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSE : LSE selected as LPUART1 clock

\_\_HAL\_RCC\_USB\_CONFIG**Description:**

- Macro to configure the USB clock (USBCLK).

**Parameters:**

- \_\_USBCLKSource: specifies the USB clock source. This parameter can be one of the following values:
  - RCC\_USBCLKSOURCE\_HSI48: HSI48 selected as USB clock
  - RCC\_USBCLKSOURCE\_PLL: PLL Clock selected as USB clock

\_\_HAL\_RCC\_GET\_USB\_SOURCE**Description:**

- Macro to get the USB clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USBCLKSOURCE\_HSI48: HSI48 selected as USB clock
  - RCC\_USBCLKSOURCE\_PLL: PLL Clock selected as USB clock

\_\_HAL\_RCC\_RNG\_CONFIG**Description:**

- Macro to configure the RNG clock (RNGCLK).

**Parameters:**

- \_\_RNGCLKSource: specifies the USB clock source. This parameter can be one of the following values:
  - RCC\_RNGCLKSOURCE\_HSI48 : HSI48 selected as RNG clock
  - RCC\_RNGCLKSOURCE\_PLLC LK: PLL Clock selected as RNG clock

\_\_HAL\_RCC\_GET\_RNG\_SOURCE**Description:**

- Macro to get the RNG clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_RNGCLKSOURCE\_HSI48 : HSI48 selected as RNG clock
  - RCC\_RNGCLKSOURCE\_PLLC LK: PLL Clock selected as RNG clock

\_\_HAL\_RCC\_HSI48M\_CONFIG**Description:**

- macro to select the HSI48M clock

source

**Parameters:**

- `__HSI48MCLKSource__`: specifies the HSI48M clock source dedicated for USB and RNG peripherals. This parameter can be one of the following values:
  - `RCC_HSI48M_PLL`: A dedicated 48MHz PLL output.
  - `RCC_HSI48M_HSI48`: 48MHz issued from internal HSI48 oscillator.

**Notes:**

- This macro can be replaced by either `__HAL_RCC_RNG_CONFIG` or `__HAL_RCC_USB_CONFIG` to configure respectively RNG or UBS clock sources.

`__HAL_RCC_GET_HSI48M_SOURCE`

**Description:**

- macro to get the HSI48M clock source.

**Return value:**

- The clock source can be one of the following values:
  - `RCC_HSI48M_PLL`: A dedicated 48MHz PLL output.
  - `RCC_HSI48M_HSI48`: 48MHz issued from internal HSI48 oscillator.

**Notes:**

- This macro can be replaced by either `__HAL_RCC_GET_RNG_SOURCE` or `__HAL_RCC_GET_USB_SOURCE` to get respectively RNG or UBS clock sources.

`__HAL_RCC_HSISTOP_ENABLE`

**Description:**

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USART and I2C.

**Return value:**

- None

**Notes:**

- The Enable of this function has not

effect on the HSION bit.

`__HAL_RCC_HSISTOP_DISABLE`  
`__HAL_RCC_LSEDRIVE_CONFIG`

**Description:**

- Macro to configures the External Low Speed oscillator (LSE) drive capability.

**Parameters:**

- `__RCC_LSEDRIVE__`: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - `RCC_LSEDRIVE_LOW`: LSE oscillator low drive capability.
  - `RCC_LSEDRIVE_MEDIUMLOW`: LSE oscillator medium low drive capability.
  - `RCC_LSEDRIVE_MEDIUMHIG`: LSE oscillator medium high drive capability.
  - `RCC_LSEDRIVE_HIGH`: LSE oscillator high drive capability.

**Return value:**

- None

`__HAL_RCC_WAKEUPSTOP_CLK_CONFIG`

**Description:**

- Macro to configures the wake up from stop clock.

**Parameters:**

- `__RCC_STOPWUCLK__`: specifies the clock source used after wake up from stop. This parameter can be one of the following values:
  - `RCC_STOP_WAKEUPCLOCK_MSI`: MSI selected as system clock source
  - `RCC_STOP_WAKEUPCLOCK_HSI`: HSI selected as system clock source

**Return value:**

- None

`__HAL_RCC_CRS_ENABLE_IT`

**Description:**

- Enables the specified CRS interrupts.

**Parameters:**

- `__INTERRUPT__`: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:

- RCC\_CRS\_IT\_SYNCOK
- RCC\_CRS\_IT\_SYNCWARN
- RCC\_CRS\_IT\_ERR
- RCC\_CRS\_IT\_ESYNC

**Return value:**

- None

[\\_\\_HAL\\_RCC\\_CRS\\_DISABLE\\_IT](#)**Description:**

- Disables the specified CRS interrupts.

**Parameters:**

- [\\_\\_INTERRUPT\\_\\_](#): specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC\_CRS\_IT\_SYNCOK
  - RCC\_CRS\_IT\_SYNCWARN
  - RCC\_CRS\_IT\_ERR
  - RCC\_CRS\_IT\_ESYNC

**Return value:**

- None

[\\_\\_HAL\\_RCC\\_CRS\\_GET\\_IT\\_SOURCE](#)**Description:**

- Check the CRS interrupt has occurred or not.

**Parameters:**

- [\\_\\_INTERRUPT\\_\\_](#): specifies the CRS interrupt source to check. This parameter can be one of the following values:
  - RCC\_CRS\_IT\_SYNCOK
  - RCC\_CRS\_IT\_SYNCWARN
  - RCC\_CRS\_IT\_ERR
  - RCC\_CRS\_IT\_ESYNC

**Return value:**

- The: new state of [\\_\\_INTERRUPT\\_\\_](#) (SET or RESET).

[RCC\\_CRS\\_IT\\_ERROR\\_MASK](#)**Description:**

- Clear the CRS interrupt pending bits bits to clear the selected interrupt pending bits.

**Parameters:**

- [\\_\\_INTERRUPT\\_\\_](#): specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC\_CRS\_IT\_SYNCOK
  - RCC\_CRS\_IT\_SYNCWARN

- RCC\_CRS\_IT\_ERR
- RCC\_CRS\_IT\_ESYNC
- RCC\_CRS\_IT\_TRIMOVF
- RCC\_CRS\_IT\_SYNCERR
- RCC\_CRS\_IT\_SYNCMISS

`_HAL_RCC_CRS_CLEAR_IT`  
`_HAL_RCC_CRS_GET_FLAG`

**Description:**

- Checks whether the specified CRS flag is set or not.

**Parameters:**

- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - RCC\_CRS\_FLAG\_SYNCOK
  - RCC\_CRS\_FLAG\_SYNCWARN
  - RCC\_CRS\_FLAG\_ERR
  - RCC\_CRS\_FLAG\_ESYNC
  - RCC\_CRS\_FLAG\_TRIMOVF
  - RCC\_CRS\_FLAG\_SYNCERR
  - RCC\_CRS\_FLAG\_SYNCMISS

**Return value:**

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

`RCC_CRS_FLAG_ERROR_MASK`

**Description:**

- Clears the CRS specified FLAG.

**Parameters:**

- `_FLAG_`: specifies the flag to clear. This parameter can be one of the following values:
  - RCC\_CRS\_FLAG\_SYNCOK
  - RCC\_CRS\_FLAG\_SYNCWARN
  - RCC\_CRS\_FLAG\_ERR
  - RCC\_CRS\_FLAG\_ESYNC
  - RCC\_CRS\_FLAG\_TRIMOVF
  - RCC\_CRS\_FLAG\_SYNCERR
  - RCC\_CRS\_FLAG\_SYNCMISS

**Return value:**

- None

`_HAL_RCC_CRS_CLEAR_FLAG`  
`_HAL_RCC_CRS_FREQ_ERROR_COUNT`  
`ER_ENABLE`

**Description:**

- Enables the oscillator clock for frequency error counter.

**Return value:**

- None

**Notes:**

- when the CEN bit is set the CRS\_CFGR register becomes write-protected.

`__HAL_RCC_CRS_FREQ_ERROR_COUNT  
ER_DISABLE`

**Description:**

- Disables the oscillator clock for frequency error counter.

**Return value:**

- None

`__HAL_RCC_CRS_AUTOMATIC_CALIB_E  
NABLE`

**Description:**

- Enables the automatic hardware adjustment of TRIM bits.

**Return value:**

- None

**Notes:**

- When the AUTOTRIMEN bit is set the CRS\_CFGR register becomes write-protected.

`__HAL_RCC_CRS_AUTOMATIC_CALIB_DI  
SABLE`

**Description:**

- Enables or disables the automatic hardware adjustment of TRIM bits.

**Return value:**

- None

`__HAL_RCC_CRS_RELOADVALUE_CALC  
ULATE`

**Description:**

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

**Parameters:**

- `__FTARGET__`: Target frequency (value in Hz)
- `__FSYNC__`: Synchronization signal frequency (value in Hz)

**Return value:**

- None

**Notes:**

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the

following: RELOAD = (fTARGET / fSYNC) -1

`_HAL_RCC_HSI_OUT_ENABLE`

**Notes:**

- After reset, the HSI output is not available

`_HAL_RCC_HSI_OUT_DISABLE`

**Notes:**

- After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable and can be used to clock the USB. The HSI48 is stopped by hardware when entering STOP and STANDBY modes.

`_HAL_RCC_HSI48_DISABLE`

`_HAL_RCC_HSI48M_DIV6_OUT_ENABLE`

**Notes:**

- After reset, the HSI48Mhz (divided by 6) output is not available

`_HAL_RCC_HSI48M_DIV6_OUT_DISABLE`

**RCC LSE CSS external interrupt line**

`RCC EXTI_LINE_LSECSS` External interrupt line 19 connected to the LSE CSS EXTI Line

**RCC HSI48M Clock Source**

`RCC_FLAG_HSI48`

`RCC_HSI48M_PLL`

`RCC_HSI48M_HSI48`

**RCC I2C1 Clock Source**

`RCC_I2C1CLKSOURCE_PCLK1`

`RCC_I2C1CLKSOURCE_SYSCLK`

`RCC_I2C1CLKSOURCE_HSI`

**RCC I2C3 Clock Source**

`RCC_I2C3CLKSOURCE_PCLK1`

`RCC_I2C3CLKSOURCE_SYSCLK`

`RCC_I2C3CLKSOURCE_HSI`

**IOPORT Peripheral Clock Enable Disable**

`_HAL_RCC_GPIOE_CLK_ENABLE`

`_HAL_RCC_GPIOE_CLK_DISABLE`

`_HAL_RCC_GPIOD_CLK_ENABLE`

`_HAL_RCC_GPIOD_CLK_DISABLE`

**IOPORT Peripheral Clock Sleep Enable Disable**

---

`_HAL_RCC_GPIOE_CLK_SLEEP_ENABLE`  
`_HAL_RCC_GPIOE_CLK_SLEEP_DISABLE`  
`_HAL_RCC_GPIOD_CLK_SLEEP_ENABLE`  
`_HAL_RCC_GPIOD_CLK_SLEEP_DISABLE`

**IOPORT Peripheral Force Release Reset**

`_HAL_RCC_GPIOE_FORCE_RESET`  
`_HAL_RCC_GPIOE_RELEASE_RESET`  
`_HAL_RCC_GPIOD_FORCE_RESET`  
`_HAL_RCC_GPIOD_RELEASE_RESET`

**RCC LPTIM1 Clock Source**

`RCC_LPTIM1CLKSOURCE_PCLK`  
`RCC_LPTIM1CLKSOURCE_LSI`  
`RCC_LPTIM1CLKSOURCE_HSI`  
`RCC_LPTIM1CLKSOURCE_LSE`

**RCC LPUART Clock Source**

`RCC_LPUART1CLKSOURCE_PCLK1`  
`RCC_LPUART1CLKSOURCE_SYSCLK`  
`RCC_LPUART1CLKSOURCE_HSI`  
`RCC_LPUART1CLKSOURCE_LSE`

**RCC LSE Drive Configuration**

`RCC_LSEDRIVE_LOW`  
`RCC_LSEDRIVE_MEDIUMLOW`  
`RCC_LSEDRIVE_MEDIUMHIGH`  
`RCC_LSEDRIVE_HIGH`

**AHB Peripheral Clock Enable Disable**

`_HAL_RCC_AES_CLK_ENABLE`  
`_HAL_RCC_AES_CLK_DISABLE`  
`_HAL_RCC_TSC_CLK_ENABLE`  
`_HAL_RCC_TSC_CLK_DISABLE`  
`_HAL_RCC RNG_CLK_ENABLE`  
`_HAL_RCC RNG_CLK_DISABLE`  
`_HAL_RCC_LSECSS_EXTI_ENABLE_IT`

**Description:**

- Enable interrupt on RCC LSE CSS EXTI Line 19.

**Return value:**

- None

**Description:**

`_HAL_RCC_LSECSS_EXTI_DISABLE_IT`

- Disable interrupt on RCC LSE CSS EXTI Line 19.

**Return value:**

- None

**Description:**

- Enable event on RCC LSE CSS EXTI Line 19.

**Return value:**

- None.

**Description:**

- Disable event on RCC LSE CSS EXTI Line 19.

**Return value:**

- None.

**Description:**

- RCC LSE CSS EXTI line configuration: set falling edge trigger.

**Return value:**

- None.

**Description:**

- Disable the RCC LSE CSS Extended Interrupt Falling Trigger.

**Return value:**

- None.

**Description:**

- RCC LSE CSS EXTI line configuration: set rising edge trigger.

**Return value:**

- None.

**Description:**

- Disable the RCC LSE CSS Extended Interrupt Rising Trigger.

**Return value:**

- None.

**Description:**

- RCC LSE CSS EXTI line

`__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT`

`__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT`

`__HAL_RCC_LSECSS_EXTI_ENABLE_FALLING_EDGE`

`__HAL_RCC_LSECSS_EXTI_DISABLE_FALLING_EDGE`

`__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_EDGE`

`__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_EDGE`

`__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_FALLING_EDGE`

configuration: set rising & falling edge trigger.

**Return value:**

- None.

`__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_FALLING_EDGE`

**Description:**

- Disable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger.

**Return value:**

- None.

`__HAL_RCC_LSECSS_EXTI_GET_FLAG`

**Description:**

- Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not.

**Return value:**

- EXTI: RCC LSE CSS Line Status.

`__HAL_RCC_LSECSS_EXTI_CLEAR_FLAG`

**Description:**

- Clear the RCC LSE CSS EXTI flag.

**Return value:**

- None.

`__HAL_RCC_LSECSS_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.

**RCC Periph Clock Selection**

`RCC_PERIPHCLK_USART1`  
`RCC_PERIPHCLK_USART2`  
`RCC_PERIPHCLK_LPUART1`  
`RCC_PERIPHCLK_I2C1`  
`RCC_PERIPHCLK_I2C2`  
`RCC_PERIPHCLK_RTC`  
`RCC_PERIPHCLK_USB`  
`RCC_PERIPHCLK_LPTIM1`  
`RCC_PERIPHCLK_LCD`  
`RCC_PERIPHCLK_I2C3`

***RCC RNG Clock Source***

RCC\_RNGCLKSOURCE\_HSI48  
RCC\_RNGCLKSOURCE\_PLLCLK

***RCC StopWakeUp Clock***

RCC\_STOP\_WAKEUPCLOCK\_MSI  
RCC\_STOP\_WAKEUPCLOCK\_HSI

***RCC TIM Prescaler Selection***

RCC\_TIMPRES\_DESACTIVATED  
RCC\_TIMPRES\_ACTIVATED

***RCC USART1 Clock Source***

RCC\_USART1CLKSOURCE\_PCLK2  
RCC\_USART1CLKSOURCE\_SYSCLK  
RCC\_USART1CLKSOURCE\_HSI  
RCC\_USART1CLKSOURCE\_LSE

***RCC USART2 Clock Source***

RCC\_USART2CLKSOURCE\_PCLK1  
RCC\_USART2CLKSOURCE\_SYSCLK  
RCC\_USART2CLKSOURCE\_HSI  
RCC\_USART2CLKSOURCE\_LSE

***RCC USB Clock Source***

RCC\_USBCLKSOURCE\_HSI48  
RCC\_USBCLKSOURCE\_PLL

## 38 HAL RNG Generic Driver

### 38.1 RNG Firmware driver registers structures

#### 38.1.1 RNG\_HandleTypeDef

##### Data Fields

- *RNG\_TypeDef \* Instance*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_RNG\_StateTypeDef State*
- *uint32\_t RandomNumber*

##### Field Documentation

- ***RNG\_TypeDef\* RNG\_HandleTypeDef::Instance***  
Register base address
- ***HAL\_LockTypeDef RNG\_HandleTypeDef::Lock***  
RNG locking object
- ***\_\_IO HAL\_RNG\_StateTypeDef RNG\_HandleTypeDef::State***  
RNG communication state
- ***uint32\_t RNG\_HandleTypeDef::RandomNumber***  
Last Generated RNG Data

### 38.2 RNG Firmware driver API description

#### 38.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using `__HAL_RCC_RNG_CLK_ENABLE()` macro. in `HAL_RNG_MspInit()`.
2. Activate the RNG peripheral using `HAL_RNG_Init()` function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using `HAL_RNG_GenerateRandomNumber()` function.

#### 38.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the `RNG_InitTypeDef` and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- [`HAL\_RNG\_Init\(\)`](#)
- [`HAL\_RNG\_DeInit\(\)`](#)

- [\*HAL\\_RNG\\_MspInit\(\)\*](#)
- [\*HAL\\_RNG\\_MspDeInit\(\)\*](#)

### 38.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- [\*HAL\\_RNG\\_GenerateRandomNumber\(\)\*](#)
- [\*HAL\\_RNG\\_GenerateRandomNumber\\_IT\(\)\*](#)
- [\*HAL\\_RNG\\_IRQHandler\(\)\*](#)
- [\*HAL\\_RNG\\_GetRandomNumber\(\)\*](#)
- [\*HAL\\_RNG\\_GetRandomNumber\\_IT\(\)\*](#)
- [\*HAL\\_RNG\\_ReadLastRandomNumber\(\)\*](#)
- [\*HAL\\_RNG\\_ReadyDataCallback\(\)\*](#)
- [\*HAL\\_RNG\\_ErrorCallback\(\)\*](#)

### 38.2.4 Peripheral State functions

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

This section contains the following APIs:

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

### 38.2.5 Detailed description of functions

#### **HAL\_RNG\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)</b>
Function Description	Initializes the RNG peripheral and creates the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>hrng:</b> pointer to a RNG_HandleTypeDef structure.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

#### **HAL\_RNG\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)</b>
Function Description	Deinitializes the RNG peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrng:</b> pointer to a RNG_HandleTypeDef structure.</li></ul>

#### **HAL\_RNG\_MspInit**

Function Name	<b>void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)</b>
Function Description	Initializes the RNG MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hrng:</b> pointer to a RNG_HandleTypeDef structure.</li></ul>

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

### HAL\_RNG\_MspDeInit

Function Name      **void HAL\_RNG\_MspDeInit (RNG\_HandleTypeDef \* hrng)**

Function Description      DeInitializes the RNG MSP.

Parameters      • **hrng:** pointer to a RNG\_HandleTypeDef structure.

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

### HAL\_RNG\_GetRandomNumber

Function Name      **uint32\_t HAL\_RNG\_GetRandomNumber (RNG\_HandleTypeDef \* hrng)**

Function Description      return generated random number in polling mode (Obsolete).

Parameters      • **hrng:** pointer to a RNG\_HandleTypeDef structure that contains the configuration information for RNG.

Return values	<ul style="list-style-type: none"> <li>• <b>random:</b> value</li> </ul>
---------------	--

### HAL\_RNG\_GetRandomNumber\_IT

Function Name      **uint32\_t HAL\_RNG\_GetRandomNumber\_IT (RNG\_HandleTypeDef \* hrng)**

Function Description      Returns a 32-bit random number with interrupt enabled (Obsolete), Use HAL\_RNG\_GenerateRandomNumber\_IT() API instead.

Parameters      • **hrng:** RNG handle

Return values	<ul style="list-style-type: none"> <li>• <b>32-bit:</b> random number</li> </ul>
---------------	--

### HAL\_RNG\_GenerateRandomNumber

Function Name      **HAL\_StatusTypeDef HAL\_RNG\_GenerateRandomNumber (RNG\_HandleTypeDef \* hrng, uint32\_t \* random32bit)**

Function Description      Generates a 32-bit random number.

Parameters      • **hrng:** pointer to a RNG\_HandleTypeDef structure.  
• **random32bit:** pointer to generated random number variable if successful.

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

Notes      • Each time the random number data is read the RNG\_FLAG\_DRDY flag is automatically cleared.

### HAL\_RNG\_GenerateRandomNumber\_IT

Function Name      **HAL\_StatusTypeDef HAL\_RNG\_GenerateRandomNumber\_IT (RNG\_HandleTypeDef \* hrng)**

Function Description      Generates a 32-bit random number in interrupt mode.

Parameters      • **hrng:** pointer to a RNG\_HandleTypeDef structure.

## Return values

- **HAL:** status

**HAL\_RNG\_ReadLastRandomNumber**

## Function Name

**uint32\_t HAL\_RNG\_ReadLastRandomNumber  
(RNG\_HandleTypeDef \* hrng)**

## Function Description

Read latest generated random number.

## Parameters

- **hrng:** pointer to a RNG\_HandleTypeDef structure.

## Return values

- **random:** value

**HAL\_RNG\_IRQHandler**

## Function Name

**void HAL\_RNG\_IRQHandler (RNG\_HandleTypeDef \* hrng)**

## Function Description

Handles RNG interrupt request.

## Parameters

- **hrng:** pointer to a RNG\_HandleTypeDef structure.

## Return values

- **None:**

## Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using \_\_HAL\_RNG\_CLEAR\_IT(). The clock error has no impact on the previously generated random numbers, and the RNG\_DR register contents can be used.
- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG\_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using \_\_HAL\_RNG\_CLEAR\_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written HAL\_RNG\_ErrorCallback() API is called once whether SEIS or CEIS are set.

**HAL\_RNG\_ErrorCallback**

## Function Name

**void HAL\_RNG\_ErrorCallback (RNG\_HandleTypeDef \* hrng)**

## Function Description

RNG error callbacks.

## Parameters

- **hrng:** pointer to a RNG\_HandleTypeDef structure.

## Return values

- **None:**

**HAL\_RNG\_ReadyDataCallback**

## Function Name

**void HAL\_RNG\_ReadyDataCallback (RNG\_HandleTypeDef \* hrng, uint32\_t random32bit)**

## Function Description

Data Ready callback in non-blocking mode.

## Parameters

- **hrng:** pointer to a RNG\_HandleTypeDef structure..
- **random32bit:** generated random value

Return values • **None:**

### **HAL\_RNG\_GetState**

Function Name	<b>HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)</b>
Function Description	Returns the RNG state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng:</b> pointer to a RNG_HandleTypeDef structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

## 38.3 RNG Firmware driver defines

### 38.3.1 RNG

#### *RNG Interrupt definition*

<b>RNG_IT_DRDY</b>	Data ready interrupt
<b>RNG_IT_CEI</b>	Clock error interrupt
<b>RNG_IT_SEI</b>	Seed error interrupt

#### *RNG Flag definition*

<b>RNG_FLAG_DRDY</b>	Data ready
<b>RNG_FLAG_CECS</b>	Clock error current status
<b>RNG_FLAG_SECS</b>	Seed error current status

#### *RNG Exported Macros*

<b>_HAL_RNG_RESET_HANDLE_STATE</b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Reset RNG handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <b>_HANDLE_</b>: RNG Handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>_HAL_RNG_ENABLE</b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Enables the RNG peripheral.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <b>_HANDLE_</b>: RNG Handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>_HAL_RNG_DISABLE</b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Disables the RNG peripheral.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <b>_HANDLE_</b>: RNG Handle</li> </ul>

**Return value:**

- None

**\_HAL\_RNG\_GET\_FLAG**

- Check the selected RNG flag status.

**Parameters:**

- \_HANDLE\_: RNG Handle
- \_FLAG\_: RNG flag This parameter can be one of the following values:
  - RNG\_FLAG\_DRDY: Data ready
  - RNG\_FLAG\_CECS: Clock error current status
  - RNG\_FLAG\_SECS: Seed error current status

**Return value:**

- The new state of \_FLAG\_ (SET or RESET).

**\_HAL\_RNG\_CLEAR\_FLAG**

- Clears the selected RNG flag status.

**Parameters:**

- \_HANDLE\_: RNG handle
- \_FLAG\_: RNG flag to clear

**Return value:**

- None

**Notes:**

- WARNING: This is a dummy macro for HAL code alignment, flags RNG\_FLAG\_DRDY, RNG\_FLAG\_CECS and RNG\_FLAG\_SECS are read-only.

**\_HAL\_RNG\_ENABLE\_IT**

- Enables the RNG interrupts.

**Parameters:**

- \_HANDLE\_: RNG Handle

**Return value:**

- None

**\_HAL\_RNG\_DISABLE\_IT**

- Disables the RNG interrupts.

**Parameters:**

- \_HANDLE\_: RNG Handle

**Return value:**

- None

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

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

**Parameters:**

- \_\_HANDLE\_\_: RNG Handle
- \_\_INTERRUPT\_\_: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG\_IT\_DRDY: Data ready interrupt
  - RNG\_IT\_CEI: Clock error interrupt
  - RNG\_IT\_SEI: Seed error interrupt

**Return value:**

- The new state of \_\_INTERRUPT\_\_ (SET or RESET).

[\\_\\_HAL\\_RNG\\_CLEAR\\_IT](#)**Description:**

- Clears the RNG interrupt status flags.

**Parameters:**

- \_\_HANDLE\_\_: RNG Handle
- \_\_INTERRUPT\_\_: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG\_IT\_CEI: Clock error interrupt
  - RNG\_IT\_SEI: Seed error interrupt

**Return value:**

- None

**Notes:**

- RNG\_IT\_DRDY flag is read-only, reading RNG\_DR register automatically clears RNG\_IT\_DRDY.

## 39 HAL RTC Generic Driver

### 39.1 RTC Firmware driver registers structures

#### 39.1.1 RTC\_InitTypeDef

##### Data Fields

- *uint32\_t HourFormat*
- *uint32\_t AsynchPrediv*
- *uint32\_t SynchPrediv*
- *uint32\_t OutPut*
- *uint32\_t OutPutRemap*
- *uint32\_t OutPutPolarity*
- *uint32\_t OutPutType*

##### Field Documentation

- ***uint32\_t RTC\_InitTypeDef::HourFormat***  
Specifies the RTC Hour Format. This parameter can be a value of [\*RTC\\_Hour\\_Formats\*](#)
- ***uint32\_t RTC\_InitTypeDef::AsynchPrediv***  
Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x7F
- ***uint32\_t RTC\_InitTypeDef::SynchPrediv***  
Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFFFF
- ***uint32\_t RTC\_InitTypeDef::OutPut***  
Specifies which signal will be routed to the RTC output. This parameter can be a value of [\*RTCEx\\_Output\\_selection\\_Definitions\*](#)
- ***uint32\_t RTC\_InitTypeDef::OutPutRemap***  
Specifies the remap for RTC output. This parameter can be a value of [\*RTC\\_Output\\_ALARM\\_OUT\\_Remap\*](#)
- ***uint32\_t RTC\_InitTypeDef::OutPutPolarity***  
Specifies the polarity of the output signal. This parameter can be a value of [\*RTC\\_Output\\_Polarity\\_Definitions\*](#)
- ***uint32\_t RTC\_InitTypeDef::OutPutType***  
Specifies the RTC Output Pin mode. This parameter can be a value of [\*RTC\\_Output\\_Type\\_ALARM\\_OUT\*](#)

#### 39.1.2 RTC\_TimeTypeDef

##### Data Fields

- *uint8\_t Hours*
- *uint8\_t Minutes*
- *uint8\_t Seconds*
- *uint8\_t TimeFormat*

- *uint32\_t SubSeconds*
- *uint32\_t SecondFraction*
- *uint32\_t DayLightSaving*
- *uint32\_t StoreOperation*

#### Field Documentation

- ***uint8\_t RTC\_TimeTypeDef::Hours***  
Specifies the RTC Time Hour. This parameter must be a number between Min\_Data = 0 and Max\_Data = 12 if the RTC\_HourFormat\_12 is selected. This parameter must be a number between Min\_Data = 0 and Max\_Data = 23 if the RTC\_HourFormat\_24 is selected
- ***uint8\_t RTC\_TimeTypeDef::Minutes***  
Specifies the RTC Time Minutes. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint8\_t RTC\_TimeTypeDef::Seconds***  
Specifies the RTC Time Seconds. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint8\_t RTC\_TimeTypeDef::TimeFormat***  
Specifies the RTC AM/PM Time. This parameter can be a value of [\*\*RTC\\_AM\\_PM\\_Definitions\*\*](#)
- ***uint32\_t RTC\_TimeTypeDef::SubSeconds***  
Specifies the RTC\_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- ***uint32\_t RTC\_TimeTypeDef::SecondFraction***  
Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV\_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL\_RTC\_GetTime function
- ***uint32\_t RTC\_TimeTypeDef::DayLightSaving***  
Specifies RTC\_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of [\*\*RTC\\_DayLightSaving\\_Definitions\*\*](#)
- ***uint32\_t RTC\_TimeTypeDef::StoreOperation***  
Specifies RTC\_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of [\*\*RTC\\_StoreOperation\\_Definitions\*\*](#)

### 39.1.3 RTC\_DateTypeDef

#### Data Fields

- *uint8\_t WeekDay*
- *uint8\_t Month*
- *uint8\_t Date*
- *uint8\_t Year*

### Field Documentation

- ***uint8\_t RTC\_DateTypeDef::WeekDay***  
Specifies the RTC Date WeekDay. This parameter can be a value of [\*\*RTC\\_WeekDay\\_Definitions\*\*](#)
- ***uint8\_t RTC\_DateTypeDef::Month***  
Specifies the RTC Date Month (in BCD format). This parameter can be a value of [\*\*RTC\\_Month\\_Date\\_Definitions\*\*](#)
- ***uint8\_t RTC\_DateTypeDef::Date***  
Specifies the RTC Date. This parameter must be a number between Min\_Data = 1 and Max\_Data = 31
- ***uint8\_t RTC\_DateTypeDef::Year***  
Specifies the RTC Date Year. This parameter must be a number between Min\_Data = 0 and Max\_Data = 99

## 39.1.4 RTC\_AlarmTypeDef

### Data Fields

- ***RTC\_TimeTypeDef AlarmTime***
- ***uint32\_t AlarmMask***
- ***uint32\_t AlarmSubSecondMask***
- ***uint32\_t AlarmDateWeekDaySel***
- ***uint8\_t AlarmDateWeekDay***
- ***uint32\_t Alarm***

### Field Documentation

- ***RTC\_TimeTypeDef RTC\_AlarmTypeDef::AlarmTime***  
Specifies the RTC Alarm Time members
- ***uint32\_t RTC\_AlarmTypeDef::AlarmMask***  
Specifies the RTC Alarm Masks. This parameter can be a value of [\*\*RTC\\_AlarmMask\\_Definitions\*\*](#)
- ***uint32\_t RTC\_AlarmTypeDef::AlarmSubSecondMask***  
Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of [\*\*RTC\\_Alarm\\_Sub\\_Seconds\\_Masks\\_Definitions\*\*](#)
- ***uint32\_t RTC\_AlarmTypeDef::AlarmDateWeekDaySel***  
Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of [\*\*RTC\\_AlarmDateWeekDay\\_Definitions\*\*](#)
- ***uint8\_t RTC\_AlarmTypeDef::AlarmDateWeekDay***  
Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of [\*\*RTC\\_WeekDay\\_Definitions\*\*](#)
- ***uint32\_t RTC\_AlarmTypeDef::Alarm***  
Specifies the alarm . This parameter can be a value of [\*\*RTC\\_Alarms\\_Definitions\*\*](#)

### 39.1.5 RTC\_HandleTypeDef

#### Data Fields

- *RTC\_TypeDef \* Instance*
- *RTC\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_RTCStateTypeDef State*

#### Field Documentation

- *RTC\_TypeDef\* RTC\_HandleTypeDef::Instance*  
Register base address
- *RTC\_InitTypeDef RTC\_HandleTypeDef::Init*  
RTC required parameters
- *HAL\_LockTypeDef RTC\_HandleTypeDef::Lock*  
RTC locking object
- *\_\_IO HAL\_RTCStateTypeDef RTC\_HandleTypeDef::State*  
Time communication state

## 39.2 RTC Firmware driver API description

### 39.2.1 Backup Domain Operating Condition

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low power modes or under reset).

### 39.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC\_CSR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

- Software reset, triggered by setting the RTCRST bit in the RCC Control Status register (RCC\_CSR).
- Power reset (BOR/POR/PDR).

### 39.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and 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()` function.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- Select the RTC clock source using the `__HAL_RCC_RTC_CONFIG()` function.
- Enable RTC Clock using the `__HAL_RCC_RTC_ENABLE()` function.

### 39.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL\_RTC\_Init() function.

#### Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL\_RTC\_SetTime() and HAL\_RTC\_SetDate() functions.
- To read the RTC Calendar, use the HAL\_RTC\_GetTime() and HAL\_RTC\_GetDate() functions.

#### Alarm configuration

- To configure the RTC Alarm use the HAL\_RTC\_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL\_RTC\_SetAlarm\_IT() function.
- To read the RTC Alarm, use the HAL\_RTC\_GetAlarm() function.

### 39.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

### 39.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
  - A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
  - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC\_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.

4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC\_TR and RTC\_DR shadow registers. The HAL\_RTC\_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

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

### 39.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- [\*HAL\\_RTC\\_SetTime\(\)\*](#)
- [\*HAL\\_RTC\\_GetTime\(\)\*](#)
- [\*HAL\\_RTC\\_SetDate\(\)\*](#)
- [\*HAL\\_RTC\\_GetDate\(\)\*](#)

### 39.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- [\*HAL\\_RTC\\_SetAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_SetAlarm\\_IT\(\)\*](#)
- [\*HAL\\_RTC\\_DeactivateAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_GetAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_AlarmIRQHandler\(\)\*](#)
- [\*HAL\\_RTC\\_AlarmAEventCallback\(\)\*](#)
- [\*HAL\\_RTC\\_PollForAlarmAEvent\(\)\*](#)

### 39.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- [\*HAL\\_RTC\\_WaitForSynchro\(\)\*](#)

### 39.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

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

### 39.2.11 Detailed description of functions

#### HAL\_RTC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initialize the RTC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### HAL\_RTC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	DeInitialize the RTC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function doesn't reset the RTC Backup Data registers.</li> </ul>

#### HAL\_RTC\_MspInit

Function Name	<b>void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initialize the RTC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### HAL\_RTC\_MspDeInit

Function Name	<b>void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	DeInitialize the RTC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### HAL\_RTC\_SetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)</b>
Function Description	Set RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sTime:</b> Pointer to Time structure</li> <li>• <b>Format:</b> Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- RTC_FORMAT_BIN: Binary data format</li> <li>- RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTC\_GetTime**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)</b>
Function Description	Get RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sTime:</b> Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.</li> <li>• <b>Format:</b> Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_FORMAT_BIN: Binary data format</li> <li>– RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S &gt;= SS</li> <li>• You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.</li> </ul>

**HAL\_RTC\_SetDate**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</b>
Function Description	Set RTC current date.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sDate:</b> Pointer to date structure</li> <li>• <b>Format:</b> specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_FORMAT_BIN: Binary data format</li> <li>– RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTC\_GetDate**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</b>
Function Description	Get RTC current date.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sDate:</b> Pointer to Date structure</li> </ul>

- **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 | • <b>HAL:</b> status  |
| Notes         | • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read. |

### **HAL\_RTC\_SetAlarm**

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTC_SetAlarm</b><br><b>(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,</b><br><b>uint32_t Format)</b>   |
| Function Description | Set the specified RTC Alarm.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sAlarm:</b> Pointer to Alarm structure</li> <li>• <b>Format:</b> Specifies the format of the entered parameters. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– RTC_FORMAT_BIN: Binary data format</li> <li>– RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>   |

### **HAL\_RTC\_SetAlarm\_IT**

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTC_SetAlarm_IT</b><br><b>(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,</b><br><b>uint32_t Format)</b>  |
| Function Description | Set the specified RTC Alarm with Interrupt.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sAlarm:</b> Pointer to Alarm structure</li> <li>• <b>Format:</b> Specifies the format of the entered parameters. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– RTC_FORMAT_BIN: Binary data format</li> <li>– RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>   |
| Notes                | <ul style="list-style-type: none"> <li>• The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).</li> <li>• The HAL_RTC_SetTime() must be called before enabling the Alarm feature.</li> </ul>  |

### **HAL\_RTC\_DeactivateAlarm**

- |               |   |
|---------------|---|
| Function Name | <b>HAL_StatusTypeDef HAL_RTC_DeactivateAlarm</b><br><b>(RTC_HandleTypeDef * hrtc, uint32_t Alarm)</b> |
|---------------|---|

Function Description	Deactivate the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>Alarm:</b> Specifies the Alarm. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_ALARM_A: AlarmA</li> <li>– RTC_ALARM_B: AlarmB</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_RTC\_GetAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetAlarm</b> <b>(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,</b> <b>uint32_t Alarm, uint32_t Format)</b>
Function Description	Get the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sAlarm:</b> Pointer to Date structure</li> <li>• <b>Alarm:</b> Specifies the Alarm. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_ALARM_A: AlarmA</li> <li>– RTC_ALARM_B: AlarmB</li> </ul> </li> <li>• <b>Format:</b> Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_FORMAT_BIN: Binary data format</li> <li>– RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_RTC\_AlarmIRQHandler

Function Name	<b>void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	Handle Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_RTC\_PollForAlarmAEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent</b> <b>(RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	Handle AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_RTC\_AlarmAEventCallback

Function Name	<b>void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Alarm A callback.

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

### HAL\_RTC\_WaitForSynchro

Function Name	<b>HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)</b>
Function Description	Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>
Notes	<ul style="list-style-type: none"> <li>The RTC Resynchronization mode is write protected, use the <code>__HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.</li> <li>To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.</li> </ul>

### HAL\_RTC\_GetState

Function Name	<b>HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)</b>
Function Description	Return the RTC handle state.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> state</li></ul>

### RTC\_EnterInitMode

Function Name	<b>HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)</b>
Function Description	Enter the RTC Initialization mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

Notes

- The RTC Initialization mode is write protected, use the `__HAL_RTC_WRITEPROTECTION_DISABLE()` before calling this function.

### RTC\_ByteToBcd2

Function Name	<b>uint8_t RTC_BYTETOBCD2 (uint8_t Value)</b>
Function Description	Convert a 2 digit decimal to BCD format.
Parameters	<ul style="list-style-type: none"><li>• <b>Value:</b> Byte to be converted</li></ul>

---

Return values	<ul style="list-style-type: none"> <li>• <b>Converted:</b> byte</li> </ul>
---------------	--

### RTC\_Bcd2ToByte

Function Name	<b>uint8_t RTC_Bcd2ToByte (uint8_t Value)</b>
Function Description	Convert from 2 digit BCD to Binary.
Parameters	<ul style="list-style-type: none"> <li>• <b>Value:</b> BCD value to be converted</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Converted:</b> word</li> </ul>

## 39.3 RTC Firmware driver defines

### 39.3.1 RTC

#### *RTC AlarmDateWeekDay Definitions*

RTC\_ALARMDATEWEEKDAYSEL\_DATE  
RTC\_ALARMDATEWEEKDAYSEL\_WEEKDAY

#### *RTC AlarmMask Definitions*

RTC\_ALARMMASK\_NONE  
RTC\_ALARMMASK\_DATEWEEKDAY  
RTC\_ALARMMASK\_HOURS  
RTC\_ALARMMASK\_MINUTES  
RTC\_ALARMMASK\_SECONDS  
RTC\_ALARMMASK\_ALL

#### *RTC Alarms Definitions*

RTC\_ALARM\_A  
RTC\_ALARM\_B

#### *RTC Alarm Sub Seconds Masks Definitions*

RTC_ALARMSUBSECONDMASK_ALL	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
RTC_ALARMSUBSECONDMASK_SS14_1	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
RTC_ALARMSUBSECONDMASK_SS14_2	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_3	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_4	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_5	SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_6	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_7	SS[14:7] are don't care in Alarm

RTC_ALARMSUBSECONDMASK_SS14_8	comparison. Only SS[6:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_9	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_10	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_11	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_12	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_13	SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
RTC_ALARMSUBSECONDMASK_SS14	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
RTC_ALARMSUBSECONDMASK_NONE	SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
	SS[14:0] are compared and must match to activate alarm.

***RTC AM PM Definitions***

RTC\_HOURFORMAT12\_AM

RTC\_HOURFORMAT12\_PM

***RTC DayLightSaving Definitions***

RTC\_DAYLIGHTSAVING\_SUB1H

RTC\_DAYLIGHTSAVING\_ADD1H

RTC\_DAYLIGHTSAVING\_NONE

***RTC Exported Macros***\_HAL\_RTC\_RESET\_HANDLE\_STATE**Description:**

- Reset RTC handle state.

**Parameters:**

- \_HANDLE\_: RTC handle.

**Return value:**

- None

**Description:**

- Disable the write protection for RTC registers.

**Parameters:**

- \_HANDLE\_: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_WRITEPROTECTION_ENABLE`

**Description:**

- Enable the write protection for RTC registers.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_ALARMA_ENABLE`

**Description:**

- Enable the RTC ALARMA peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_ALARMA_DISABLE`

**Description:**

- Disable the RTC ALARMA peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_ALARM_B_ENABLE`

**Description:**

- Enable the RTC ALARM\_B peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_ALARM_B_DISABLE`

**Description:**

- Disable the RTC ALARM\_B peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Enable the RTC Alarm interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC\_IT\_ALRA: Alarm A interrupt
  - RTC\_IT\_ALRB: Alarm B interrupt

**Return value:**

- None

**Description:**

- Disable the RTC Alarm interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC\_IT\_ALRA: Alarm A interrupt
  - RTC\_IT\_ALRB: Alarm B interrupt

**Return value:**

- None

**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
  - [RTC\\_IT\\_ALRB](#): Alarm B 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 check. This parameter can be:
  - [RTC\\_IT\\_ALRA](#): Alarm A interrupt
  - [RTC\\_IT\\_ALRB](#): Alarm B interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_ALARM\\_GET\\_FLAG](#)**Description:**

- Get the selected RTC Alarm's flag status.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the RTC handle.
- [\\_\\_FLAG\\_\\_](#): specifies the RTC Alarm Flag sources to check. This parameter can be:
  - [RTC\\_FLAG\\_ALRAF](#)
  - [RTC\\_FLAG\\_ALRBF](#)
  - [RTC\\_FLAG\\_ALRAWF](#)
  - [RTC\\_FLAG\\_ALRBWF](#)

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_CLEAR\_FLAG****Description:**

- Clear the RTC Alarm's pending flags.

**Parameters:**

- \_HANDLE\_: specifies the RTC handle.
- \_FLAG\_: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC\_FLAG\_ALRAF
  - RTC\_FLAG\_ALRBF

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_IT****Description:**

- Enable interrupt on the RTC Alarm associated Exti line.

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_IT****Description:**

- Disable interrupt on the RTC Alarm associated Exti line.

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_EVENT****Description:**

- Enable event on the RTC Alarm associated Exti line.

**Return value:**

- None.

**\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_EVENT****Description:**

- Disable event on the RTC Alarm associated Exti line.

**Return value:**

- None.

**\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_FALLING\_EDGE****Description:**

- Enable falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Disable falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Enable rising edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

**Description:**

- Clear the RTC Alarm

associated Exti line flag.

**Return value:**

- None.

`__HAL_RTC_ALARM_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on RTC Alarm associated Exti line.

**Return value:**

- None.

***RTC Flags Definitions***

`RTC_FLAG_RECALPF`  
`RTC_FLAG_TAMP2F`  
`RTC_FLAG_TAMP1F`  
`RTC_FLAG_TSOVF`  
`RTC_FLAG_TSF`  
`RTC_FLAG_WUTF`  
`RTC_FLAG_ALRBF`  
`RTC_FLAG_ALRAF`  
`RTC_FLAG_INITF`  
`RTC_FLAG_RSF`  
`RTC_FLAG_INITS`  
`RTC_FLAG_SHPF`  
`RTC_FLAG_WUTWF`  
`RTC_FLAG_ALRBWF`  
`RTC_FLAG_ALRAWF`

***RTC Hour Formats***

`RTC_HOURFORMAT_24`  
`RTC_HOURFORMAT_12`

***RTC Input Parameter Format Definitions***

`RTC_FORMAT_BIN`  
`RTC_FORMAT_BCD`

***RTC Interrupts Definitions***

<code>RTC_IT_TS</code>	Enable Timestamp Interrupt
<code>RTC_IT_WUT</code>	Enable Wakeup timer Interrupt
<code>RTC_IT_ALRA</code>	Enable Alarm A Interrupt
<code>RTC_IT_ALRB</code>	Enable Alarm B Interrupt
<code>RTC_IT_TAMP</code>	Enable all Tamper Interrupt

RTC\_IT\_TAMP1      Enable Tamper 1 Interrupt  
RTC\_IT\_TAMP2      Enable Tamper 2 Interrupt

***RTC Private macros to check input parameters***

IS\_RTC\_HOUR\_FORMAT  
IS\_RTC\_OUTPUT\_POL  
IS\_RTC\_OUTPUT\_TYPE  
IS\_RTC\_OUTPUT\_REMAP  
IS\_RTC\_HOURFORMAT12  
IS\_RTC\_DAYLIGHT\_SAVING  
IS\_RTC\_STORE\_OPERATION  
IS\_RTC\_FORMAT  
IS\_RTC\_YEAR  
IS\_RTC\_MONTH  
IS\_RTC\_DATE  
IS\_RTC\_WEEKDAY  
IS\_RTC\_ALARM\_DATE\_WEEKDAY\_DATE  
IS\_RTC\_ALARM\_DATE\_WEEKDAY\_WEEKDAY  
IS\_RTC\_ALARM\_DATE\_WEEKDAY\_SEL  
IS\_RTC\_ALARM\_MASK  
IS\_RTC\_ALARM  
IS\_RTC\_ALARM\_SUB\_SECOND\_VALUE  
IS\_RTC\_ALARM\_SUB\_SECOND\_MASK  
IS\_RTC\_ASYNCH\_PREDIV  
IS\_RTC\_SYNCH\_PREDIV  
IS\_RTC\_HOUR12  
IS\_RTC\_HOUR24  
IS\_RTC\_MINUTES  
IS\_RTC\_SECONDS

***RTC Month Date Definitions***

RTC\_MONTH\_JANUARY  
RTC\_MONTH\_FEBRUARY  
RTC\_MONTH\_MARCH  
RTC\_MONTH\_APRIIL  
RTC\_MONTH\_MAY  
RTC\_MONTH\_JUNE  
RTC\_MONTH\_JULY

RTC\_MONTH\_AUGUST  
RTC\_MONTH\_SEPTEMBER  
RTC\_MONTH\_OCTOBER  
RTC\_MONTH\_NOVEMBER  
RTC\_MONTH\_DECEMBER

***RTC Output ALARM OUT Remap***

RTC\_OUTPUT\_REMAP\_NONE  
RTC\_OUTPUT\_REMAP\_POS1

***RTC Output Polarity Definitions***

RTC\_OUTPUT\_POLARITY\_HIGH  
RTC\_OUTPUT\_POLARITY\_LOW

***RTC Output Type ALARM OUT***

RTC\_OUTPUT\_TYPE\_OPENDRAIN  
RTC\_OUTPUT\_TYPE\_PUSHPULL

***RTC StoreOperation Definitions***

RTC\_STOREOPERATION\_RESET  
RTC\_STOREOPERATION\_SET

***RTC WeekDay Definitions***

RTC\_WEEKDAY\_MONDAY  
RTC\_WEEKDAY\_TUESDAY  
RTC\_WEEKDAY\_WEDNESDAY  
RTC\_WEEKDAY\_THURSDAY  
RTC\_WEEKDAY\_FRIDAY  
RTC\_WEEKDAY\_SATURDAY  
RTC\_WEEKDAY\_SUNDAY

## 40 HAL RTC Extension Driver

### 40.1 RTCEx Firmware driver registers structures

#### 40.1.1 RTC\_TamperTypeDef

##### Data Fields

- *uint32\_t Tamper*
- *uint32\_t Interrupt*
- *uint32\_t Trigger*
- *uint32\_t NoErase*
- *uint32\_t MaskFlag*
- *uint32\_t Filter*
- *uint32\_t SamplingFrequency*
- *uint32\_t PrechargeDuration*
- *uint32\_t TamperPullUp*
- *uint32\_t TimeStampOnTamperDetection*

##### Field Documentation

- ***uint32\_t RTC\_TamperTypeDef::Tamper***  
Specifies the Tamper Pin. This parameter can be a value of  
*RTCEx\_Tamper\_Pins\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::Interrupt***  
Specifies the Tamper Interrupt. This parameter can be a value of  
*RTCEx\_Tamper\_Interrupt\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::Trigger***  
Specifies the Tamper Trigger. This parameter can be a value of  
*RTCEx\_Tamper\_Trigger\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::NoErase***  
Specifies the Tamper no erase mode. This parameter can be a value of  
*RTCEx\_Tamper\_EraseBackUp\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::MaskFlag***  
Specifies the Tamper Flag masking. This parameter can be a value of  
*RTCEx\_Tamper\_MaskFlag\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::Filter***  
Specifies the RTC Filter Tamper. This parameter can be a value of  
*RTCEx\_Tamper\_Filter\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::SamplingFrequency***  
Specifies the sampling frequency. This parameter can be a value of  
*RTCEx\_Tamper\_Sampling\_Frequencies\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::PrechargeDuration***  
Specifies the Precharge Duration . This parameter can be a value of  
*RTCEx\_Tamper\_Pin\_Precharge\_Duration\_Definitions*
- ***uint32\_t RTC\_TamperTypeDef::TamperPullUp***  
Specifies the Tamper PullUp . This parameter can be a value of  
*RTCEx\_Tamper\_Pull\_UP\_Definitions*

- ***uint32\_t RTC\_TamperTypeDef::TimeStampOnTamperDetection***  
Specifies the TimeStampOnTamperDetection. This parameter can be a value of [\*RTCEx\\_Tamper\\_TimeStampOnTamperDetection\\_Definitions\*](#)

## 40.2 RTCEx Firmware driver API description

### 40.2.1 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- [\*HAL\\_RTCEx\\_SetTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetTimeStamp\\_IT\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEx\\_GetTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetTamper\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetTamper\\_IT\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateTamper\(\)\*](#)
- [\*HAL\\_RTCEx\\_TamperTimeStampIRQHandler\(\)\*](#)
- [\*HAL\\_RTCEx\\_TimeStampEventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_Tamper1EventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_Tamper2EventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_Tamper3EventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTimeStampEvent\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTamper1Event\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTamper2Event\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTamper3Event\(\)\*](#)

### 40.2.2 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- [\*HAL\\_RTCEx\\_SetWakeUpTimer\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetWakeUpTimer\\_IT\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateWakeUpTimer\(\)\*](#)
- [\*HAL\\_RTCEx\\_GetWakeUpTimer\(\)\*](#)
- [\*HAL\\_RTCEx\\_WakeUpTimerIRQHandler\(\)\*](#)
- [\*HAL\\_RTCEx\\_WakeUpTimerEventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForWakeUpTimerEvent\(\)\*](#)

### 40.2.3 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).

- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- [\*HAL\\_RTCEx\\_BKUPWrite\(\)\*](#)
- [\*HAL\\_RTCEx\\_BKUPRead\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetSmoothCalib\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetSynchroShift\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetCalibrationOutPut\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateCalibrationOutPut\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetRefClock\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateRefClock\(\)\*](#)
- [\*HAL\\_RTCEx\\_EnableBypassShadow\(\)\*](#)
- [\*HAL\\_RTCEx\\_DisableBypassShadow\(\)\*](#)

#### 40.2.4 Extended features functions

This section provides functions allowing to:

- RTC Alram B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- [\*HAL\\_RTCEx\\_AlarmBEventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForAlarmBEvent\(\)\*](#)

#### 40.2.5 Detailed description of functions

##### **HAL\_RTCEx\_SetTimeStamp**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</b>
Function Description	SetTimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>TimeStampEdge:</b> Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.</li> <li>– RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.</li> </ul> </li> <li>• <b>RTC_TimeStampPin:</b> specifies the RTC TimeStamp Pin. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin on STM32L05x/6x/7x/8x and PA2 on STM32L03x/4x/2x/1x.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

**HAL\_RTCEx\_SetTimeStamp\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</b>
Function Description	SetTimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>TimeStampEdge:</b> Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.</li> <li>– RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.</li> </ul> </li> <li>• <b>RTC_TimeStampPin:</b> Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin on STM32L05x/6x/7x/8x and PA2 on STM32L03x/4x/2x/1x.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

**HAL\_RTCEx\_DeactivateTimeStamp**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)</b>
Function Description	Deactivate TimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_GetTimeStamp**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)</b>
Function Description	Get the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sTimeStamp:</b> Pointer to Time structure</li> <li>• <b>sTimeStampDate:</b> Pointer to Date structure</li> <li>• <b>Format:</b> specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– RTC_FORMAT_BIN: Binary data format</li> <li>– RTC_FORMAT_BCD: BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_SetTamper**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</b>
Function Description	Set Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sTamper:</b> Pointer to Tamper Structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we disable the tamper interrupt for all tampers.</li> </ul>

**HAL\_RTCEx\_SetTamper\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</b>
Function Description	Set Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>sTamper:</b> Pointer to RTC Tamper.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we force the tamper interrupt for all tampers.</li> </ul>

**HAL\_RTCEx\_DeactivateTamper**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)</b>
Function Description	Deactivate Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>Tamper:</b> Selected tamper pin. This parameter can be RTC_TAMPER_1 and/or RTC_TAMPER_2 for STM32L05x/6x. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3 for STM32L01x/2x/3x/7x/8x.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_TamperTimeStampIRQHandler**

Function Name	<b>void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	Handle TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_RTCEx\_Tamper1EventCallback**

Function Name	<b>void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)</b>
---------------	---

Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_RTCEx\_Tamper2EventCallback

Function Name	<b>void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_RTCEx\_Tamper3EventCallback

Function Name	<b>void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_RTCEx\_TimeStampEventCallback

Function Name	<b>void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_RTCEx\_PollForTimeStampEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	Handle TimeStamp polling request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li><li>• <b>Timeout:</b> Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_RTCEx\_PollForTamper1Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	Handle Tamper 1 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc:</b> RTC handle</li><li>• <b>Timeout:</b> Timeout duration</li></ul>

Return values • **HAL:** status

### **HAL\_RTCEx\_PollForTamper2Event**

Function Name **HAL\_StatusTypeDef HAL\_RTCEx\_PollForTamper2Event  
(RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

Function Description Handle Tamper 2 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values • **HAL:** status

### **HAL\_RTCEx\_PollForTamper3Event**

Function Name **HAL\_StatusTypeDef HAL\_RTCEx\_PollForTamper3Event  
(RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

Function Description Handle Tamper 3 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values • **HAL:** status

### **HAL\_RTCEx\_SetWakeUpTimer**

Function Name **HAL\_StatusTypeDef HAL\_RTCEx\_SetWakeUpTimer  
(RTC\_HandleTypeDef \* hrtc, uint32\_t WakeUpCounter,  
uint32\_t WakeUpClock)**

Function Description Set wake up timer.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values • **HAL:** status

### **HAL\_RTCEx\_SetWakeUpTimer\_IT**

Function Name **HAL\_StatusTypeDef HAL\_RTCEx\_SetWakeUpTimer\_IT  
(RTC\_HandleTypeDef \* hrtc, uint32\_t WakeUpCounter,  
uint32\_t WakeUpClock)**

Function Description Set wake up timer with interrupt.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values • **HAL:** status

### **HAL\_RTCEx\_DeactivateWakeUpTimer**

Function Name **uint32\_t HAL\_RTCEx\_DeactivateWakeUpTimer  
(RTC\_HandleTypeDef \* hrtc)**

Function Description	Deactivate wake up timer counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_GetWakeUpTimer**

Function Name	<b>uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)</b>
Function Description	Get wake up timer counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Counter:</b> value</li> </ul>

**HAL\_RTCEx\_WakeUpTimerIRQHandler**

Function Name	<b>void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	Handle Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_RTCEx\_WakeUpTimerEventCallback**

Function Name	<b>void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_RTCEx\_PollForWakeUpTimerEvent**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	Handle Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_BKUPWrite**

Function Name	<b>void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)</b>
Function Description	Write a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>BackupRegister:</b> RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.</li> </ul>

- **Data:** Data to be written in the specified RTC Backup data register.
  - **None:**
- Return values

**HAL\_RTCEx\_BKUPRead**

Function Name	<code>uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)</code>
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>BackupRegister:</b> RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Read:</b> value</li> </ul>

**HAL\_RTCEx\_SetSmoothCalib**

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)</code>
Function Description	Set the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>SmoothCalibPeriod:</b> Select the Smooth Calibration Period. This parameter can be can be one of the following values : <ul style="list-style-type: none"> <li>- RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.</li> <li>- RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.</li> <li>- RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.</li> </ul> </li> <li>• <b>SmoothCalibPlusPulses:</b> Select to Set or reset the CALP bit. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.</li> <li>- RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.</li> </ul> </li> <li>• <b>SmoothCalibMinusPulsesValue:</b> Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue mut be equal to 0.</li> </ul>

**HAL\_RTCEx\_SetSynchroShift**

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t</code>
---------------	---

**ShiftSubFS**

Function Description	Configure the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>ShiftAdd1S:</b> Select to add or not 1 second to the time calendar. This parameter can be one of the following values : <ul style="list-style-type: none"> <li>- RTC_SHIFTADD1S_SET: Add one second to the clock calendar.</li> <li>- RTC_SHIFTADD1S_RESET: No effect.</li> </ul> </li> <li>• <b>ShiftSubFS:</b> Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When REFCKON is set, firmware must not write to Shift control register.</li> </ul>

**HAL\_RTCEx\_SetCalibrationOutPut**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)</b>
Function Description	Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>CalibOutput:</b> : Select the Calibration output Selection . This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.</li> <li>- RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_DeactivateCalibrationOutPut**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)</b>
Function Description	Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_SetRefClock**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)</b>
Function Description	Enable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_RTCEx\_DeactivateRefClock**

Function Name      **HAL\_StatusTypeDef HAL\_RTCEx\_DeactivateRefClock  
(RTC\_HandleTypeDef \* hrtc)**

Function Description      Disable the RTC reference clock detection.

Parameters      • **hrtc:** RTC handle

Return values      • **HAL:** status

**HAL\_RTCEx\_EnableBypassShadow**

Function Name      **HAL\_StatusTypeDef HAL\_RTCEx\_EnableBypassShadow  
(RTC\_HandleTypeDef \* hrtc)**

Function Description      Enable the Bypass Shadow feature.

Parameters      • **hrtc:** RTC handle

Return values      • **HAL:** status

Notes      • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

**HAL\_RTCEx\_DisableBypassShadow**

Function Name      **HAL\_StatusTypeDef HAL\_RTCEx\_DisableBypassShadow  
(RTC\_HandleTypeDef \* hrtc)**

Function Description      Disable the Bypass Shadow feature.

Parameters      • **hrtc:** RTC handle

Return values      • **HAL:** status

Notes      • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

**HAL\_RTCEx\_AlarmBEventCallback**

Function Name      **void HAL\_RTCEx\_AlarmBEventCallback  
(RTC\_HandleTypeDef \* hrtc)**

Function Description      Alarm B callback.

Parameters      • **hrtc:** RTC handle

Return values      • **None:**

**HAL\_RTCEx\_PollForAlarmBEvent**

Function Name      **HAL\_StatusTypeDef HAL\_RTCEx\_PollForAlarmBEvent  
(RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

Function Description      Handle Alarm B Polling request.

Parameters      • **hrtc:** RTC handle

                  • **Timeout:** Timeout duration

Return values      • **HAL:** status

## 40.3 RTCEx Firmware driver defines

### 40.3.1 RTCEx

#### *RTCEx Add 1 Second Parameter Definitions*

RTC\_SHIFTADD1S\_RESET

RTC\_SHIFTADD1S\_SET

#### *RTCEx Backup Registers Definition*

RTC\_BKP\_DR0

RTC\_BKP\_DR1

RTC\_BKP\_DR2

RTC\_BKP\_DR3

RTC\_BKP\_DR4

#### *RTC Calibration*

`__HAL_RTC_CALIBRATION_OUTPUT_ENABLE`

##### **Description:**

- Enable the RTC calibration output.

##### **Parameters:**

- `__HANDLE__`: specifies the RTC handle.

##### **Return value:**

- None

`__HAL_RTC_CALIBRATION_OUTPUT_DISABLE`

##### **Description:**

- Disable the calibration output.

##### **Parameters:**

- `__HANDLE__`: specifies the RTC handle.

##### **Return value:**

- None

`__HAL_RTC_CLOCKREF_DETECTION_ENABLE`

##### **Description:**

- Enable the clock reference detection.

##### **Parameters:**

- `__HANDLE__`: specifies the RTC handle.

##### **Return value:**

- None

`__HAL_RTC_CLOCKREF_DETECTION_DISABLE`

##### **Description:**

- Disable the clock reference

detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Get the selected RTC shift operation's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - `RTC_FLAG_SHPF`

**Return value:**

- None

***RTCEEx Calib Output selection Definitions***

`RTC_CALIBOUTPUT_512HZ`

`RTC_CALIBOUTPUT_1HZ`

***RTCEEx Flags Definitions***

`RTC_FLAG_TAMP3F`

***RTCEEx Interrupts Definitions***

`RTC_IT_TAMP3`

***Private macros to check input parameters***

`IS_RTC_OUTPUT`

`IS_RTC_BKP`

`IS_TIMESTAMP_EDGE`

`IS_RTC_TAMPER`

`IS_RTC_TAMPER_INTERRUPT`

`IS_RTC_TIMESTAMP_PIN`

`IS_RTC_TAMPER_TRIGGER`

`IS_RTC_TAMPER_ERASE_MODE`

`IS_RTC_TAMPER_MASKFLAG_STATE`

`IS_RTC_TAMPER_FILTER`

`IS_RTC_TAMPER_SAMPLING_FREQ`

`IS_RTC_TAMPER_PRECHARGE_DURATION`

IS\_RTC\_TAMPER\_TIMESTAMPON TAMPER\_DETECTION

IS\_RTC\_TAMPER\_PULLUP\_STATE

IS\_RTC\_WAKEUP\_CLOCK

IS\_RTC\_WAKEUP\_COUNTER

IS\_RTC\_SMOOTH\_CALIB\_PERIOD

IS\_RTC\_SMOOTH\_CALIB\_PLUS

IS\_RTC\_SHIFT\_ADD1S

IS\_RTC\_CALIB\_OUTPUT

#### ***RTCEx Output Selection Definition***

RTC\_OUTPUT\_DISABLE

RTC\_OUTPUT\_ALARMA

RTC\_OUTPUT\_ALARMB

RTC\_OUTPUT\_WAKEUP

#### ***RTCEx Smooth calib Minus pulses Definitions***

IS\_RTC\_SMOOTH\_CALIB\_MINUS

#### ***RTCEx Smooth calib period Definitions***

RTC\_SMOOTHCALIB\_PERIOD\_32SEC If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC\_SMOOTHCALIB\_PERIOD\_16SEC If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC\_SMOOTHCALIB\_PERIOD\_8SEC If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

#### ***RTCEx Smooth calib Plus pulses Definitions***

RTC\_SMOOTHCALIB\_PLUSPULSES\_SET The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC\_SMOOTHCALIB\_PLUSPULSES\_RESET The number of RTCCLK pulses substituted during a 32-second window = CALM[8:0]

#### ***RTCEx Subtract Fraction Of Second Value***

IS\_RTC\_SHIFT\_SUBFS

#### ***RTC Tamper***

\_\_HAL\_RTC\_TAMPER1\_ENABLE

#### **Description:**

- Enable the RTC Tamper1 input detection.

#### **Parameters:**

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

#### **Return value:**

- None

`__HAL_RTC_TAMPER1_DISABLE`

**Description:**

- Disable the RTC Tamper1 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_TAMPER2_ENABLE`

**Description:**

- Enable the RTC Tamper2 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_TAMPER2_DISABLE`

**Description:**

- Disable the RTC Tamper2 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_TAMPER3_ENABLE`

**Description:**

- Enable the RTC Tamper3 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_TAMPER3_DISABLE`

**Description:**

- Disable the RTC Tamper3 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**\_HAL\_RTC\_TAMPER\_ENABLE\_IT**

- 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\_TAMP: All tampers interrupts
  - RTC\_IT\_TAMP1: Tamper1 interrupt
  - RTC\_IT\_TAMP2: Tamper2 interrupt
  - RTC\_IT\_TAMP3: Tamper3 interrupt

**Return value:**

- None

**\_HAL\_RTC\_TAMPER\_DISABLE\_IT**

- 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\_TAMP: All tampers interrupts
  - RTC\_IT\_TAMP1: Tamper1 interrupt
  - RTC\_IT\_TAMP2: Tamper2 interrupt
  - RTC\_IT\_TAMP3: Tamper3 interrupt

**Return value:**

- None

**\_HAL\_RTC\_TAMPER\_GET\_IT**

- Check whether the specified RTC Tamper interrupt has occurred or not.

**Parameters:**

- HANDLE: specifies the RTC handle.

- \_\_INTERRUPT\_\_: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC\_IT\_TAMP1: Tamper1 interrupt
  - RTC\_IT\_TAMP2: Tamper2 interrupt
  - RTC\_IT\_TAMP3: Tamper3 interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_TAMPER\\_GET\\_IT\\_SOURCE](#)**Description:**

- Check whether the specified RTC Tamper interrupt has been enabled or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC\_IT\_TAMP: All tampers interrupts
  - RTC\_IT\_TAMP1: Tamper1 interrupt
  - RTC\_IT\_TAMP2: Tamper2 interrupt
  - RTC\_IT\_TAMP3: Tamper3 interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_TAMPER\\_GET\\_FLAG](#)**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC\_FLAG\_TAMP1F: Tamper1 flag
  - RTC\_FLAG\_TAMP2F: Tamper2 flag
  - RTC\_FLAG\_TAMP3F: Tamper3 flag

**Return value:**

---

**\_HAL\_RTC\_TAMPER\_CLEAR\_FLAG**

- None

**Description:**

- Clear the RTC Tamper's pending flags.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC\_FLAG\_TAMP1F: Tamper1 flag
  - RTC\_FLAG\_TAMP2F: Tamper2 flag
  - RTC\_FLAG\_TAMP3F: Tamper3 flag

**Return value:**

- None

***RTCEEx Tamper EraseBackUp Definitions***

RTC\_TAMPER\_ERASE\_BACKUP\_ENABLE

RTC\_TAMPER\_ERASE\_BACKUP\_DISABLE

***RTCEEx Tamper Filter Definitions***

RTC\_TAMPERFILTER\_DISABLE Tamper filter is disabled

RTC\_TAMPERFILTER\_2SAMPLE Tamper is activated after 2 consecutive samples at the active level

RTC\_TAMPERFILTER\_4SAMPLE Tamper is activated after 4 consecutive samples at the active level

RTC\_TAMPERFILTER\_8SAMPLE Tamper is activated after 8 consecutive samples at the active leve.

***RTCEEx Tamper Interrupt Definitions***

RTC\_TAMPER1\_INTERRUPT

RTC\_TAMPER2\_INTERRUPT

RTC\_TAMPER3\_INTERRUPT

RTC\_ALL\_TAMPER\_INTERRUPT

***RTCEEx Tamper MaskFlag Definitions***

RTC\_TAMPERMASK\_FLAG\_DISABLE

RTC\_TAMPERMASK\_FLAG\_ENABLE

***RTCEEx Tamper Pins Definition***

RTC\_TAMPER\_1

RTC\_TAMPER\_2

RTC\_TAMPER\_3

***RTCEEx Tamper Pin Precharge Duration Definitions***

RTC_TAMPERPRECHARGEDURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
RTC_TAMPERPRECHARGEDURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

***RTCEx Tamper Pull UP Definitions***

RTC_TAMPER_PULLUP_ENABLE	Tamper pins are pre-charged before sampling
RTC_TAMPER_PULLUP_DISABLE	Tamper pins pre-charge is disabled

***RTCEx Tamper Sampling Frequencies Definitions***

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

***EXTI RTC Tamper Timestamp EXTI***

<u>__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_I</u>	<b>Description:</b>
T	<ul style="list-style-type: none"> <li>Enable interrupt on the RTC Tamper and Timestamp associated Exti line.</li> </ul>

**Return value:**

- None

**Description:**

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None

**Description:**

- Enable event on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Disable event on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

**Return value:**

- None.

**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

***RTCEx TamperTimeStampOnTamperDetection Definitions***

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TIMESTAMPONTAMPERDETECTION_DISABLE	TimeStamp on Tamper Detection event is not saved

***RTCEx Tamper Trigger Definitions***

RTC_TAMPERTRIGGER_RISINGEDGE
RTC_TAMPERTRIGGER_FALLINGEDGE
RTC_TAMPERTRIGGER_LOWLEVEL
RTC_TAMPERTRIGGER_HIGHLEVEL

***RTC Timestamp***\_HAL\_RTC\_TIMESTAMP\_ENABLE**Description:**

- Enable the RTC TimeStamp peripheral.

**Parameters:**

- \_HANDLE\_: specifies the RTC handle.

**Return value:**

- None

\_HAL\_RTC\_TIMESTAMP\_DISABLE**Description:**

- Disable the RTC TimeStamp peripheral.

**Parameters:**

- \_HANDLE\_: specifies the RTC handle.

**Return value:**

- None

\_HAL\_RTC\_TIMESTAMP\_ENABLE\_IT**Description:**

- Enable the RTC TimeStamp interrupt.

**Parameters:**

- \_HANDLE\_: specifies the RTC handle.
- \_INTERRUPT\_: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  - RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_DISABLE_IT`

**Description:**

- Disable the RTC TimeStamp interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  - `RTC_IT_TS`: TimeStamp interrupt

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_GET_IT`

**Description:**

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt to check. This parameter can be:
  - `RTC_IT_TS`: TimeStamp interrupt

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_GET_IT_SOURCE`

**Description:**

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - `RTC_IT_TS`: TimeStamp interrupt

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_GET_FLAG`

**Description:**

- Get the selected RTC TimeStamp's flag status.

**Parameters:**

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  - RTC\_FLAG\_TSF
  - RTC\_FLAG\_TSOVF

**Return value:**

- None

\_HAL\_RTC\_TIMESTAMP\_CLEAR\_FLAG**Description:**

- Clear the RTC Time Stamp's pending flags.

**Parameters:**

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC Alarm Flag to clear. This parameter can be:
  - RTC\_FLAG\_TSF

**Return value:**

- None

***RTCEx TimeStamp Pin Selection***RTC\_TIMESTAMPIN\_DEFAULT***RTCEx Time Stamp Edges definition***RTC\_TIMESTAMPEDGE\_RISINGRTC\_TIMESTAMPEDGE\_FALLING***RTC WakeUp Timer***\_HAL\_RTC\_WAKEUPTIMER\_ENABLE**Description:**

- Enable the RTC WakeUp Timer peripheral.

**Parameters:**

- HANDLE: specifies the RTC handle.

**Return value:**

- None

\_HAL\_RTC\_WAKEUPTIMER\_DISABLE**Description:**

- Disable the RTC WakeUp Timer peripheral.

**Parameters:**

- HANDLE: specifies the RTC

handle.

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_ENABLE_IT`

- Enable the RTC WakeUpTimer interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer interrupt

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_DISABLE_IT`

- Disable the RTC WakeUpTimer interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer interrupt

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_GET_IT`

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer interrupt

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE`

**Description:**

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer interrupt

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_GET_FLAG`

**Description:**

- Get the selected RTC WakeUpTimer's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
  - `RTC_FLAG_WUTF`
  - `RTC_FLAG_WUTWF`

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG`

**Description:**

- Clear the RTC Wake Up timer's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - `RTC_FLAG_WUTF`

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_INTERRUPT`

**Description:**

- Enable interrupt on the RTC WakeUp Timer associated Exti

line.

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT`

- Description:**
- Disable interrupt on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT`

- Description:**
- Enable event on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT`

- Description:**
- Disable event on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE`

- Description:**
- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE`

- Description:**
- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE`

- Description:**
- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE`

- Description:**
- Disable rising edge trigger on the RTC WakeUp Timer associated

Exti line.

**Return value:**

- None.

**Description:**

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

**Description:**

- Clear the RTC WakeUp Timer associated Exti line flag.

**Return value:**

- None.

**Description:**

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

***RTCEx Wakeup Timer Definitions***

RTC\_WAKEUPCLOCK\_RTCCLK\_DIV16  
RTC\_WAKEUPCLOCK\_RTCCLK\_DIV8  
RTC\_WAKEUPCLOCK\_RTCCLK\_DIV4  
RTC\_WAKEUPCLOCK\_RTCCLK\_DIV2  
RTC\_WAKEUPCLOCK\_CK\_SPRE\_16BITS  
RTC\_WAKEUPCLOCK\_CK\_SPRE\_17BITS

## 41 HAL SMARTCARD Generic Driver

### 41.1 SMARTCARD Firmware driver registers structures

#### 41.1.1 SMARTCARD\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*
- *uint32\_t OneBitSampling*
- *uint32\_t Prescaler*
- *uint32\_t GuardTime*
- *uint32\_t NACKState*
- *uint32\_t TimeOutEnable*
- *uint32\_t TimeOutValue*
- *uint32\_t BlockLength*
- *uint32\_t AutoRetryCount*

##### Field Documentation

- ***uint32\_t SMARTCARD\_InitTypeDef::BaudRate***  
Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hsc->Init.BaudRate)))
- ***uint32\_t SMARTCARD\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD\_Word\_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32\_t SMARTCARD\_InitTypeDef::StopBits***  
Specifies the number of stop bits ***SMARTCARD\_Stop\_Bits***. Only 0.5 or 1.5 stop bits are authorized in SmartCard mode.
- ***uint32\_t SMARTCARD\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of ***SMARTCARD\_Parity***  
**Note:**The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint32\_t SMARTCARD\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD\_Mode***
- ***uint32\_t SMARTCARD\_InitTypeDef::CLKPolarity***  
Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD\_Clock\_Polarity***
- ***uint32\_t SMARTCARD\_InitTypeDef::CLKPhase***  
Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD\_Clock\_Phase***

- **`uint32_t SMARTCARD_InitTypeDef::CLKLastBit`**  
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [\*\*SMARTCARD\\_Last\\_Bit\*\*](#)
- **`uint32_t SMARTCARD_InitTypeDef::OneBitSampling`**  
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [\*\*SMARTCARD\\_OneBit\\_Sampling\*\*](#)
- **`uint32_t SMARTCARD_InitTypeDef::Prescaler`**  
Specifies the SmartCard Prescaler
- **`uint32_t SMARTCARD_InitTypeDef::GuardTime`**  
Specifies the SmartCard Guard Time
- **`uint32_t SMARTCARD_InitTypeDef::NACKState`**  
Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of [\*\*SMARTCARD\\_NACK\\_Enable\*\*](#)
- **`uint32_t SMARTCARD_InitTypeDef::TimeOutEnable`**  
Specifies whether the receiver timeout is enabled. This parameter can be a value of [\*\*SMARTCARD\\_Timeout\\_Enable\*\*](#)
- **`uint32_t SMARTCARD_InitTypeDef::TimeOutValue`**  
Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- **`uint32_t SMARTCARD_InitTypeDef::BlockLength`**  
Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- **`uint32_t SMARTCARD_InitTypeDef::AutoRetryCount`**  
Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

#### 41.1.2 SMARTCARD\_AdvFeatureInitTypeDef

##### Data Fields

- **`uint32_t AdvFeatureInit`**
- **`uint32_t TxPinLevelInvert`**
- **`uint32_t RxPinLevelInvert`**
- **`uint32_t DataInvert`**
- **`uint32_t Swap`**
- **`uint32_t OverrunDisable`**
- **`uint32_t DMADisableonRxError`**
- **`uint32_t MSBFirst`**

##### Field Documentation

- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit`**  
Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of [\*\*SMARTCARD\\_Advanced\\_Features\\_Initialization\\_Type\*\*](#)
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert`**  
Specifies whether the TX pin active level is inverted. This parameter can be a value of [\*\*SMARTCARD\\_Tx\\_Inv\*\*](#)

- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert`**  
Specifies whether the RX pin active level is inverted. This parameter can be a value of **`SMARTCARD_Rx_Inv`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert`**  
Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of **`SMARTCARD_Data_Inv`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap`**  
Specifies whether TX and RX pins are swapped. This parameter can be a value of **`SMARTCARD_Rx_Tx_Swap`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable`**  
Specifies whether the reception overrun detection is disabled. This parameter can be a value of **`SMARTCARD_Overrun_Disable`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError`**  
Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of **`SMARTCARD_DMA_Disable_on_Rx_Error`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst`**  
Specifies whether MSB is sent first on UART line. This parameter can be a value of **`SMARTCARD_MSB_First`**

#### 41.1.3 SMARTCARD\_HandleTypeDef

##### Data Fields

- **`USART_TypeDef * Instance`**
- **`SMARTCARD_InitTypeDef Init`**
- **`SMARTCARD_AdvFeatureInitTypeDef AdvancedInit`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`uint16_t RxXferCount`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_SMARTCARD_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

##### Field Documentation

- **`USART_TypeDef* SMARTCARD_HandleTypeDef::Instance`**
- **`SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init`**
- **`SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit`**
- **`uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr`**
- **`uint16_t SMARTCARD_HandleTypeDef::TxXferSize`**
- **`uint16_t SMARTCARD_HandleTypeDef::TxXferCount`**
- **`uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr`**
- **`uint16_t SMARTCARD_HandleTypeDef::RxXferSize`**
- **`uint16_t SMARTCARD_HandleTypeDef::RxXferCount`**
- **`DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx`**

- `DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock`
- `_IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State`
- `_IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode`

## 41.2 SMARTCARD Firmware driver API description

### 41.2.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity.
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The `HAL_SMARTCARD_Init()` API follow respectively the USART (a)synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- `HAL_SMARTCARD_Init()`
- `HAL_SMARTCARD_DelInit()`
- `HAL_SMARTCARD_MspInit()`
- `HAL_SMARTCARD_MspDelInit()`

### 41.2.2 IO operation functions

This section contains the following APIs:

- `HAL_SMARTCARD_Transmit()`
- `HAL_SMARTCARD_Receive()`
- `HAL_SMARTCARD_Transmit_IT()`
- `HAL_SMARTCARD_Receive_IT()`
- `HAL_SMARTCARD_Transmit_DMA()`
- `HAL_SMARTCARD_Receive_DMA()`
- `HAL_SMARTCARD_IRQHandler()`
- `HAL_SMARTCARD_TxCpltCallback()`
- `HAL_SMARTCARD_RxCpltCallback()`
- `HAL_SMARTCARD_ErrorCallback()`

### 41.2.3 Peripheral State functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL\_SMARTCARD\_GetState() API is helpful to check in run-time the state of the SMARTCARD peripheral
- SMARTCARD\_SetConfig() API configures the SMARTCARD peripheral
- SMARTCARD\_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization

This section contains the following APIs:

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

### 41.2.4 Detailed description of functions

#### **HAL\_SMARTCARD\_Init**

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

#### **HAL\_SMARTCARD\_DelInit**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_DelInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Delinitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### **HAL\_SMARTCARD\_MspInit**

Function Name	<b>void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

#### **HAL\_SMARTCARD\_MspDelInit**

Function Name	<b>void HAL_SMARTCARD_MspDelInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> </ul>

- Return values • **None:**

### **HAL\_SMARTCARD\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> <li>• <b>Timeout:</b> : Timeout duration</li> </ul>
Return values	• <b>HAL:</b> status

### **HAL\_SMARTCARD\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> <li>• <b>Timeout:</b> : Timeout duration</li> </ul>
Return values	• <b>HAL:</b> status

### **HAL\_SMARTCARD\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	• <b>HAL:</b> status

### **HAL\_SMARTCARD\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> </ul>
Return values	• <b>HAL:</b> status

**HAL\_SMARTCARD\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SMARTCARD\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> </ul>

**HAL\_SMARTCARD\_IRQHandler**

Function Name	<b>void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD interrupt requests handling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMARTCARD\_TxCpltCallback**

Function Name	<b>void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMARTCARD\_RxCpltCallback**

Function Name	<b>void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> </ul>

Return values • **None:**

### **HAL\_SMARTCARD\_ErrorCallback**

Function Name **void HAL\_SMARTCARD\_ErrorCallback(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description SMARTCARD error callbacks.

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:** SMARTCARD handle

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

## **41.3 SMARTCARD Firmware driver defines**

### **41.3.1 SMARTCARD**

#### ***SMARTCARD Advanced Features Initialization***

**SMARTCARD\_ADVFEATURE\_NO\_INIT**

**SMARTCARD\_ADVFEATURE\_TXINVERT\_INIT**

**SMARTCARD\_ADVFEATURE\_RXINVERT\_INIT**

**SMARTCARD\_ADVFEATURE\_DATAINVERT\_INIT**

**SMARTCARD\_ADVFEATURE\_SWAP\_INIT**

**SMARTCARD\_ADVFEATURE\_RXOVERRUNDISABLE\_INIT**

**SMARTCARD\_ADVFEATURE\_DMADISABLEONERROR\_INIT**

**SMARTCARD\_ADVFEATURE\_MSBFIRST\_INIT**

**IS\_SMARTCARD\_ADVFEATURE\_INIT**

#### ***SMARTCARD Clock Phase***

SMARTCARD\_PHASE\_1EDGE  
SMARTCARD\_PHASE\_2EDGE  
IS\_SMARTCARD\_PHASE  
**SMARTCARD Clock Polarity**  
SMARTCARD\_POLARITY\_LOW  
SMARTCARD\_POLARITY\_HIGH  
IS\_SMARTCARD\_POLARITY  
**SMARTCARD CR3 LSB Position**  
SMARTCARD\_CR3\_SCARCNT\_LSB\_POS  
**SMARTCARD Data Inv**  
SMARTCARD\_ADVFEATURE\_DATAINV\_DISABLE  
SMARTCARD\_ADVFEATURE\_DATAINV\_ENABLE  
IS\_SMARTCARD\_ADVFEATURE\_DATAINV  
**SMARTCARD DMA on Rx Error**  
SMARTCARD\_ADVFEATURE\_DMA\_ENABLEONRXERROR  
SMARTCARD\_ADVFEATURE\_DMA\_DISABLEONRXERROR  
IS\_SMARTCARD\_ADVFEATURE\_DMAONRXERROR  
**SMARTCARD DMA Requests**  
SMARTCARD\_DMAREQ\_TX  
SMARTCARD\_DMAREQ\_RX  
**SMARTCARD Error Code**  
HAL\_SMARTCARD\_ERROR\_NONE    No error  
HAL\_SMARTCARD\_ERROR\_PE    Parity error  
HAL\_SMARTCARD\_ERROR\_NE    Noise error  
HAL\_SMARTCARD\_ERROR\_FE    frame error  
HAL\_SMARTCARD\_ERROR\_ORE    Overrun error  
HAL\_SMARTCARD\_ERROR\_DMA    DMA transfer error  
HAL\_SMARTCARD\_ERROR\_RTO    Receiver TimeOut error  
**SMARTCARD Exported Macros**  
**\_\_HAL\_SMARTCARD\_RESET\_HANDLE\_STATE**    **Description:**  
•    Reset SMARTCARD handle state.  
**Parameters:**  
•    \_\_HANDLE\_\_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2  
**Return value:**

- None

### \_\_HAL\_SMARTCARD\_FLUSH\_DRREGISTER

**Description:**

- Flushes the Smartcard DR register.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle.

**Return value:**

- None

### \_\_HAL\_SMARTCARD\_CLEAR\_FLAG

**Description:**

- Clears the specified SMARTCARD pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD\_CLEAR\_PEF
  - SMARTCARD\_CLEAR\_FEF
  - SMARTCARD\_CLEAR\_NEF
  - SMARTCARD\_CLEAR\_OREF
  - SMARTCARD\_CLEAR\_IDLEF
  - SMARTCARD\_CLEAR\_TCF
  - SMARTCARD\_CLEAR\_RTOF
  - SMARTCARD\_CLEAR\_EOBF

**Return value:**

- None

### \_\_HAL\_SMARTCARD\_CLEAR\_PEFLAG

**Description:**

- Clear the SMARTCARD PE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle.

**Return value:**

- None

### \_\_HAL\_SMARTCARD\_CLEAR\_FEFLAG

**Description:**

- Clear the SMARTCARD FE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle.

**Return value:**

- None

[\\_\\_HAL\\_SMARTCARD\\_CLEAR\\_NEFLAG](#)**Description:**

- Clear the SMARTCARD NE pending flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle.

**Return value:**

- None

[\\_\\_HAL\\_SMARTCARD\\_CLEAR\\_OREFLAG](#)**Description:**

- Clear the SMARTCARD ORE pending flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle.

**Return value:**

- None

[\\_\\_HAL\\_SMARTCARD\\_CLEAR\\_IDLEFLAG](#)**Description:**

- Clear the SMARTCARD IDLE pending flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle.

**Return value:**

- None

[\\_\\_HAL\\_SMARTCARD\\_GET\\_FLAG](#)**Description:**

- Checks whether the specified Smartcard flag is set or not.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- [\\_\\_FLAG\\_\\_](#): specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD\_FLAG\_RXACK: Receive enable acknowledge flag
  - SMARTCARD\_FLAG\_TEACK: Transmit enable acknowledge flag
  - SMARTCARD\_FLAG\_BUSY: Busy flag
  - SMARTCARD\_FLAG\_EOBF: End of block flag
  - SMARTCARD\_FLAG\_RTOF: Receiver timeout flag

- SMARTCARD\_FLAG\_TXE: Transmit data register empty flag
- SMARTCARD\_FLAG\_TC: Transmission Complete flag
- SMARTCARD\_FLAG\_RXNE: Receive data register not empty flag
- SMARTCARD\_FLAG\_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\_ENABLE\_IT****Description:**

- Enables the specified SmartCard interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- \_\_INTERRUPT\_\_: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOB: End Of Block interrupt
  - SMARTCARD\_IT\_RTO: Receive TimeOut interrupt
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

[\\_\\_HAL\\_SMARTCARD\\_DISABLE\\_IT](#)**Description:**

- Disables the specified SmartCard interrupt.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- [\\_\\_INTERRUPT\\_\\_](#): specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOB: End Of Block interrupt
  - SMARTCARD\_IT\_RTO: Receive TimeOut interrupt
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

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

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

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- [\\_\\_IT\\_\\_](#): specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOB: End Of Block interrupt
  - SMARTCARD\_IT\_RTO: Receive TimeOut interrupt
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt

- SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
- SMARTCARD\_IT\_IDLE: Idle line detection interrupt
- SMARTCARD\_IT\_ORE: OverRun Error interrupt
- SMARTCARD\_IT\_NE: Noise Error interrupt
- SMARTCARD\_IT\_FE: Framing Error interrupt
- SMARTCARD\_IT\_PE: Parity Error interrupt

**Return value:**

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

`_HAL_SMARTCARD_GET_IT_SOURCE`

E

**Description:**

- Checks whether the specified SmartCard interrupt interrupt source is enabled.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `_IT_`: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOB: End Of Block interrupt
  - SMARTCARD\_IT\_RTO: Receive TimeOut interrupt
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_ORE: OverRun Error interrupt
  - SMARTCARD\_IT\_NE: Noise Error interrupt
  - SMARTCARD\_IT\_FE: Framing Error interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt

**Return value:**

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

[\\_\\_HAL\\_SMARTCARD\\_CLEAR\\_IT](#)**Description:**

- Clears the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- [\\_\\_IT\\_CLEAR\\_\\_](#): specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - SMARTCARD\_CLEAR\_PEF: Parity error clear flag
  - SMARTCARD\_CLEAR\_FEF: Framing error clear flag
  - SMARTCARD\_CLEAR\_NEF: Noise detected clear flag
  - SMARTCARD\_CLEAR\_OREF: OverRun error clear flag
  - SMARTCARD\_CLEAR\_IDLEF: Idle line detection clear flag
  - SMARTCARD\_CLEAR\_TCF: Transmission complete clear flag
  - SMARTCARD\_CLEAR\_RTOF: Receiver timeout clear flag
  - SMARTCARD\_CLEAR\_EOBF: End of block clear flag

**Return value:**

- None

[\\_\\_HAL\\_SMARTCARD\\_SEND\\_REQ](#)**Description:**

- Set a specific SMARTCARD request flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- [\\_\\_REQ\\_\\_](#): specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD\_RXDATA\_FLUSH\_R EQUEST: Receive Data flush Request
  - SMARTCARD\_TXDATA\_FLUSH\_R EQUEST: Transmit data flush Request

**Return value:**

- None

`_HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE`

**Description:**

- Enables the SMARTCARD one bit sample method.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle.

**Return value:**

- None

`_HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE`

**Description:**

- Disables the SMARTCARD one bit sample method.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle.

**Return value:**

- None

`_HAL_SMARTCARD_ENABLE`

**Description:**

- Enable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

`_HAL_SMARTCARD_DISABLE`

**Description:**

- Disable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

`_HAL_SMARTCARD_DMA_REQUEST_ENABLE`

**Description:**

- Macros to enable or disable the SmartCard DMA request.

**Parameters:**

- HANDLE: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- REQUEST: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - SMARTCARD\_DMAREQ\_TX: SmartCard DMA transmit request
  - SMARTCARD\_DMAREQ\_RX: SmartCard DMA receive request

\_HAL\_SMARTCARD\_DMA\_REQUEST\_DISABLEIS\_SMARTCARD\_BAUDRATE**Description:**

- Check the Baud rate range.

**Parameters:**

- BAUDRATE: Baud rate set by the configuration function.

**Return value:**

- Test: result (TRUE or FALSE)

IS\_SMARTCARD\_BLOCKLENGTH**Description:**

- Check the block length range.

**Parameters:**

- LENGTH: block length.

**Return value:**

- Test: result (TRUE or FALSE)

IS\_SMARTCARD\_TIMEOUT\_VALUE**Description:**

- Check the receiver timeout value.

**Parameters:**

- TIMEOUTVALUE: receiver timeout value.

**Return value:**

- Test: result (TRUE or FALSE)

IS\_SMARTCARD\_AUTORETRY\_COUN  
T**Description:**

- Check the SMARTCARD autoretry counter value.

**Parameters:**

- COUNT: number of retransmissions

**Return value:**

- Test: result (TRUE or FALSE)

***SMARTCARD Flags***

SMARTCARD_FLAG_RXACK	SMARTCARD receive enable acknowledge flag
SMARTCARD_FLAG_TEACK	SMARTCARD transmit enable acknowledge flag
SMARTCARD_FLAG_BUSY	SMARTCARD busy flag
SMARTCARD_FLAG_EOBF	SMARTCARD end of block flag
SMARTCARD_FLAG_RTOF	SMARTCARD receiver timeout flag
SMARTCARD_FLAG_TXE	SMARTCARD transmit data register empty
SMARTCARD_FLAG_TC	SMARTCARD transmission complete
SMARTCARD_FLAG_RXNE	SMARTCARD read data register not empty
SMARTCARD_FLAG_IDLE	SMARTCARD idle line detection
SMARTCARD_FLAG_ORE	SMARTCARD overrun error
SMARTCARD_FLAG_NE	SMARTCARD noise error
SMARTCARD_FLAG_FE	SMARTCARD frame error
SMARTCARD_FLAG_PE	SMARTCARD parity error

***SMARTCARD GTPR GT LSB Position***

SMARTCARD\_GTPR\_GT\_LSB\_POS

***SMARTCARD Interruption Mask***

SMARTCARD\_IT\_MASK

***SMARTCARD Interrupt definition***

SMARTCARD_IT_PE	SMARTCARD parity error interruption
SMARTCARD_IT_TXE	SMARTCARD transmit data register empty interruption
SMARTCARD_IT_TC	SMARTCARD transmission complete interruption
SMARTCARD_IT_RXNE	SMARTCARD read data register not empty interruption
SMARTCARD_IT_IDLE	SMARTCARD idle line detection interruption
SMARTCARD_IT_ERR	SMARTCARD error interruption
SMARTCARD_IT_ORE	SMARTCARD overrun error interruption
SMARTCARD_IT_NE	SMARTCARD noise error interruption
SMARTCARD_IT_FE	SMARTCARD frame error interruption
SMARTCARD_IT_EOB	SMARTCARD end of block interruption
SMARTCARD_IT_RTO	SMARTCARD receiver timeout interruption

***SMARTCARD IT CLEAR Flags***

SMARTCARD_CLEAR_PEF	Parity Error Clear Flag
SMARTCARD_CLEAR_FEF	Framing Error Clear Flag
SMARTCARD_CLEAR_NEF	Noise detected Clear Flag
SMARTCARD_CLEAR_OREF	OverRun Error Clear Flag
SMARTCARD_CLEAR_IDLEF	IDLE line detected Clear Flag

SMARTCARD\_CLEAR\_TCF      Transmission Complete Clear Flag

SMARTCARD\_CLEAR\_RTOF      Receiver Time Out Clear Flag

SMARTCARD\_CLEAR\_EOBF      End Of Block Clear Flag

***SMARTCARD Last Bit***

SMARTCARD\_LASTBIT\_DISABLE

SMARTCARD\_LASTBIT\_ENABLE

IS\_SMARTCARD\_LASTBIT

***SMARTCARD Mode***

SMARTCARD\_MODE\_RX

SMARTCARD\_MODE\_TX

SMARTCARD\_MODE\_TX\_RX

IS\_SMARTCARD\_MODE

***SMARTCARD MSB First***

SMARTCARD\_ADVFEATURE\_MSBFIRST\_DISABLE

SMARTCARD\_ADVFEATURE\_MSBFIRST\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_MSBFIRST

***SMARTCARD NACK Enable***

SMARTCARD\_NACK\_ENABLE

SMARTCARD\_NACK\_DISABLE

IS\_SMARTCARD\_NACK

***SMARTCARD OneBit Sampling***

SMARTCARD\_ONE\_BIT\_SAMPLE\_DISABLE

SMARTCARD\_ONE\_BIT\_SAMPLE\_ENABLE

IS\_SMARTCARD\_ONE\_BIT\_SAMPLE

***SMARTCARD Overrun Enabling***

SMARTCARD\_ADVFEATURE\_OVERRUN\_ENABLE

SMARTCARD\_ADVFEATURE\_OVERRUN\_DISABLE

IS\_SMARTCARD\_OVERRUN

***SMARTCARD Parity***

SMARTCARD\_PARITY\_EVEN

SMARTCARD\_PARITY\_ODD

IS\_SMARTCARD\_PARITY

***SMARTCARD Request Parameters***

SMARTCARD\_RXDATA\_FLUSH\_REQUEST      Receive Data flush Request

SMARTCARD\_TXDATA\_FLUSH\_REQUEST      Transmit data flush Request

IS\_SMARTCARD\_REQUEST\_PARAMETER

***SMARTCARD RTOR BLEN LSB Position***

SMARTCARD\_RTOR\_BLEN\_LSB\_POS

**SMARTCARD Rx Inv**

SMARTCARD\_ADVFEATURE\_RXINV\_DISABLE

SMARTCARD\_ADVFEATURE\_RXINV\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_RXINV

**SMARTCARD Rx Tx Swap**

SMARTCARD\_ADVFEATURE\_SWAP\_DISABLE

SMARTCARD\_ADVFEATURE\_SWAP\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_SWAP

**SMARTCARD Stop Bits**

SMARTCARD\_STOPBITS\_0\_5

SMARTCARD\_STOPBITS\_1\_5

IS\_SMARTCARD\_STOPBITS

**SMARTCARD Timeout Enable**

SMARTCARD\_TIMEOUT\_DISABLE

SMARTCARD\_TIMEOUT\_ENABLE

IS\_SMARTCARD\_TIMEOUT

**SMARTCARD Tx Inv**

SMARTCARD\_ADVFEATURE\_TXINV\_DISABLE

SMARTCARD\_ADVFEATURE\_TXINV\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_TXINV

**SMARTCARD Word Length**

SMARTCARD\_WORDLENGTH\_9B

IS\_SMARTCARD\_WORD\_LENGTH

## 42 HAL SMARTCARD Extension Driver

### 42.1 SMARTCARDEX Firmware driver API description

#### 42.1.1 How to use this driver

The Extended SMARTCARD HAL driver can be used as follow:

1. After having configured the SMARTCARD basic features with `HAL_SMARTCARD_Init()`, then if required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsc AdvancedInit structure.

#### 42.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- `HAL_SMARTCARDEX_BlockLength_Config()` API allows to configure the Block Length on the fly
- `HAL_SMARTCARDEX_TimeOut_Config()` API allows to configure the receiver timeout value on the fly
- `HAL_SMARTCARDEX_EnableReceiverTimeOut()` API enables the receiver timeout feature
- `HAL_SMARTCARDEX_DisableReceiverTimeOut()` API disables the receiver timeout feature

This section contains the following APIs:

- [`HAL\_SMARTCARDEX\_BlockLength\_Config\(\)`](#)
- [`HAL\_SMARTCARDEX\_TimeOut\_Config\(\)`](#)
- [`HAL\_SMARTCARDEX\_EnableReceiverTimeOut\(\)`](#)
- [`HAL\_SMARTCARDEX\_DisableReceiverTimeOut\(\)`](#)

#### 42.1.3 Detailed description of functions

##### `HAL_SMARTCARDEX_BlockLength_Config`

Function Name	<code>void HAL_SMARTCARDEX_BlockLength_Config(SMARTCARD_HandleTypeDef * hsc, uint8_t BlockLength)</code>
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>BlockLength:</b> SMARTCARD block length (8-bit long at most)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

##### `HAL_SMARTCARDEX_TimeOut_Config`

Function Name	<code>void HAL_SMARTCARDEX_TimeOut_Config(SMARTCARD_HandleTypeDef * hsc, uint32_t TimeOutValue)</code>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>TimeOutValue:</b> receiver timeout value in number of baud blocks. The timeout value must be less or equal to</li> </ul>

0xFFFFFFFF.

Return values

- **None:**

### **HAL\_SMARTCARDEX\_EnableReceiverTimeOut**

Function Name

**HAL\_StatusTypeDef  
HAL\_SMARTCARDEX\_EnableReceiverTimeOut  
(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsc:** SMARTCARD handle

Return values

- **HAL:** status

### **HAL\_SMARTCARDEX\_DisableReceiverTimeOut**

Function Name

**HAL\_StatusTypeDef  
HAL\_SMARTCARDEX\_DisableReceiverTimeOut  
(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsc:** SMARTCARD handle

Return values

- **HAL:** status

## **42.2 SMARTCARDEX Firmware driver defines**

### **42.2.1 SMARTCARDEX**

#### ***SMARTCARDEX Exported Macros***

**SMARTCARD\_GETCLOCKSOURCE Description:**

- Reports the SMARTCARD clock source.

#### **Parameters:**

- **\_\_HANDLE\_\_:** specifies the USART Handle
- **\_\_CLOCKSOURCE\_\_:** : output variable

#### **Return value:**

- **the:** USART clocking source, written in **\_\_CLOCKSOURCE\_\_**.

## 43 HAL SMBUS Generic Driver

### 43.1 SMBUS Firmware driver registers structures

#### 43.1.1 SMBUS\_InitTypeDef

##### Data Fields

- *uint32\_t Timing*
- *uint32\_t AnalogFilter*
- *uint32\_t OwnAddress1*
- *uint32\_t AddressingMode*
- *uint32\_t DualAddressMode*
- *uint32\_t OwnAddress2*
- *uint32\_t OwnAddress2Masks*
- *uint32\_t GeneralCallMode*
- *uint32\_t NoStretchMode*
- *uint32\_t PacketErrorCheckMode*
- *uint32\_t PeripheralMode*
- *uint32\_t SMBusTimeout*

##### Field Documentation

- ***uint32\_t SMBUS\_InitTypeDef::Timing***  
Specifies the SMBUS\_TIMINGR\_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32\_t SMBUS\_InitTypeDef::AnalogFilter***  
Specifies if Analog Filter is enable or not. This parameter can be a value of [\*\*SMBUS\\_Analog\\_Filter\*\*](#)
- ***uint32\_t SMBUS\_InitTypeDef::OwnAddress1***  
Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32\_t SMBUS\_InitTypeDef::AddressingMode***  
Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of [\*\*SMBUS\\_addressing\\_mode\*\*](#)
- ***uint32\_t SMBUS\_InitTypeDef::DualAddressMode***  
Specifies if dual addressing mode is selected. This parameter can be a value of [\*\*SMBUS\\_dual\\_addressing\\_mode\*\*](#)
- ***uint32\_t SMBUS\_InitTypeDef::OwnAddress2***  
Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- ***uint32\_t SMBUS\_InitTypeDef::OwnAddress2Masks***  
Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of [\*\*SMBUS\\_own\\_address2\\_masks\*\*](#)
- ***uint32\_t SMBUS\_InitTypeDef::GeneralCallMode***  
Specifies if general call mode is selected. This parameter can be a value of [\*\*SMBUS\\_general\\_call\\_addressing\\_mode\*\*](#)

- ***uint32\_t SMBUS\_InitTypeDef::NoStretchMode***  
Specifies if nostretch mode is selected. This parameter can be a value of **SMBUS\_nostretch\_mode**
- ***uint32\_t SMBUS\_InitTypeDef::PacketErrorCheckMode***  
Specifies if Packet Error Check mode is selected. This parameter can be a value of **SMBUS\_packet\_error\_check\_mode**
- ***uint32\_t SMBUS\_InitTypeDef::PeripheralMode***  
Specifies which mode of Periphal is selected. This parameter can be a value of **SMBUS\_peripheral\_mode**
- ***uint32\_t SMBUS\_InitTypeDef::SMBusTimeout***  
Specifies the content of the 32 Bits SMBUS\_TIMEOUT\_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

### 43.1.2 SMBUS\_HandleTypeDef

#### Data Fields

- ***I2C\_TypeDef \* Instance***
- ***SMBUS\_InitTypeDef Init***
- ***uint8\_t \* pBuffPtr***
- ***uint16\_t XferSize***
- ***\_\_IO uint16\_t XferCount***
- ***\_\_IO uint32\_t XferOptions***
- ***\_\_IO uint32\_t PreviousState***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO uint32\_t State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***I2C\_TypeDef\* SMBUS\_HandleTypeDef::Instance***  
SMBUS registers base address
- ***SMBUS\_InitTypeDef SMBUS\_HandleTypeDef::Init***  
SMBUS communication parameters
- ***uint8\_t\* SMBUS\_HandleTypeDef::pBuffPtr***  
Pointer to SMBUS transfer buffer
- ***uint16\_t SMBUS\_HandleTypeDef::XferSize***  
SMBUS transfer size
- ***\_\_IO uint16\_t SMBUS\_HandleTypeDef::XferCount***  
SMBUS transfer counter
- ***\_\_IO uint32\_t SMBUS\_HandleTypeDef::XferOptions***  
SMBUS transfer options
- ***\_\_IO uint32\_t SMBUS\_HandleTypeDef::PreviousState***  
SMBUS communication Previous tate
- ***HAL\_LockTypeDef SMBUS\_HandleTypeDef::Lock***  
SMBUS locking object
- ***\_\_IO uint32\_t SMBUS\_HandleTypeDef::State***  
SMBUS communication state
- ***\_\_IO uint32\_t SMBUS\_HandleTypeDef::ErrorCode***  
SMBUS Error code , see SMBUS\_Error\_Code

## 43.2 SMBUS Firmware driver API description

### 43.2.1 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SMBUSx peripheral:

- User must implement HAL\_SMBUS\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC ).
- Call the function HAL\_SMBUS\_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filter mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL\_SMBUS\_DelInit() to restore the default configuration of the selected SMBUSx peripheral.

This section contains the following APIs:

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

### 43.2.2 IO operation functions

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

1. Blocking mode function to check if device is ready for usage is :
  - HAL\_SMBUS\_IsDeviceReady()
2. There is only one mode of transfer:
  - No-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. No-Blocking mode functions with Interrupt are :
  - HAL\_SMBUS\_Master\_Transmit\_IT()
  - HAL\_SMBUS\_Master\_Receive\_IT()
  - HAL\_SMBUS\_Slave\_Transmit\_IT()
  - HAL\_SMBUS\_Slave\_Receive\_IT()
  - HAL\_SMBUS\_EnableListen\_IT()
  - HAL\_SMBUS\_EnableAlert\_IT()
  - HAL\_SMBUS\_DisableAlert\_IT()
4. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_SMBUS\_MasterTxCpltCallback()
  - HAL\_SMBUS\_MasterRxCpltCallback()

- HAL\_SMBUS\_SlaveTxCpltCallback()
- HAL\_SMBUS\_SlaveRxCpltCallback()
- HAL\_SMBUS\_AddrCallback()
- HAL\_SMBUS\_ListenCpltCallback()
- HAL\_SMBUS\_ErrorCallback()

This section contains the following APIs:

- [\*\*\*HAL\\_SMBUS\\_Master\\_Transmit\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_Master\\_Receive\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_Master\\_Abort\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_Slave\\_Transmit\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_Slave\\_Receive\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_EnableListen\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_DisableListen\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_EnableAlert\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_DisableAlert\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_IsDeviceReady\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_EV\\_IRQHandler\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_ER\\_IRQHandler\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_MasterTxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_MasterRxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_SlaveTxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_SlaveRxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_AddrCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_ListenCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMBUS\\_ErrorCallback\(\)\*\*\*](#)

### 43.2.3 Peripheral State and Errors functions

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

This section contains the following APIs:

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

### 43.2.4 Detailed description of functions

#### HAL\_SMBUS\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Initializes the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### HAL\_SMBUS\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_DelInit (SMBUS_HandleTypeDef * hsmbus)</b>
---------------	---

Function Description	Deinitializes the SMBUS peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SMBUS\_MspInit**

Function Name	<b>void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	SMBUS MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_MspDeInit**

Function Name	<b>void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	SMBUS MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_EnableAlert\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Enable SMBUS alert.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SMBUS\_DisableAlert\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Disable SMBUS alert.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SMBUS\_EnableListen\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function enable the Address listen mode in Slave mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SMBUS\_DisableListen\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function disable the Address listen mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SMBUS\_IsDeviceReady**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)</b>
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>DevAddress:</b> Target device address</li> <li>• <b>Trials:</b> Number of trials</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function is used with Memory devices</li> </ul>

**HAL\_SMBUS\_Master\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Transmit in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>DevAddress:</b> Target device address</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>

- **XferOptions:** Options of Transfer, value of SMBUS Transfer Request Definition
  - **HAL:** status
- Return values

### **HAL\_SMBUS\_Master\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Receive in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>DevAddress:</b> Target device address</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> <li>• <b>XferOptions:</b> Options of Transfer, value of SMBUS Transfer Request Definition</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_SMBUS\_Master\_Abort\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)</b>
Function Description	Abort a master/host SMBUS process communication with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>DevAddress:</b> Target device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

Notes

- : This abort can be called only if state is ready

### **HAL\_SMBUS\_Slave\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Transmit in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> <li>• <b>XferOptions:</b> Options of Transfer, value of SMBUS Transfer Request Definition</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
<b>HAL_SMBUS_Slave_Receive_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Receive in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> <li>• <b>XferOptions:</b> Options of Transfer, value of SMBUS Transfer Request Definition</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
<b>HAL_SMBUS_EV_IRQHandler</b>	
Function Name	<b>void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function handles SMBUS event interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
<b>HAL_SMBUS_ER_IRQHandler</b>	
Function Name	<b>void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function handles SMBUS error interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
<b>HAL_SMBUS_MasterTxCpltCallback</b>	
Function Name	<b>void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_MasterRxCpltCallback**

Function Name	<b>void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_SlaveTxCpltCallback**

Function Name	<b>void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_SlaveRxCpltCallback**

Function Name	<b>void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_AddrCallback**

Function Name	<b>void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)</b>
Function Description	Slave Address Match callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>TransferDirection:</b> Master request Transfer Direction (Write/Read)</li> <li>• <b>AddrMatchCode:</b> Address Match Code</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_ListenCpltCallback**

Function Name	<b>void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)</b>
---------------	---

Function Description	Slave Listen Complete callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_ErrorCallback**

Function Name	<b>void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	SMBUS error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SMBUS\_GetState**

Function Name	<b>uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Returns the SMBUS state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus:</b> : SMBUS handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

**HAL\_SMBUS\_GetError**

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

## 43.3 SMBUS Firmware driver defines

### 43.3.1 SMBUS

***SMBUS Addressing Mode***

SMBUS\_ADDRESSINGMODE\_7BIT  
 SMBUS\_ADDRESSINGMODE\_10BIT  
 IS\_SMBUS\_ADDRESSING\_MODE

***SMBUS Analog Filter***

SMBUS\_ANALOGFILTER\_ENABLE  
 SMBUS\_ANALOGFILTER\_DISABLE

`IS_SMBUS_ANALOG_FILTER`

**SMBUS Dual Addressing Mode**

`SMBUS_DUALADDRESS_DISABLE`

`SMBUS_DUALADDRESS_ENABLE`

`IS_SMBUS_DUAL_ADDRESS`

**SMBUS Error Code**

<code>HAL_SMBUS_ERROR_NONE</code>	No error
-----------------------------------	----------

<code>HAL_SMBUS_ERROR_BERR</code>	BERR error
-----------------------------------	------------

<code>HAL_SMBUS_ERROR_ARLO</code>	ARLO error
-----------------------------------	------------

<code>HAL_SMBUS_ERROR_ACKF</code>	ACKF error
-----------------------------------	------------

<code>HAL_SMBUS_ERROR_OVR</code>	OVR error
----------------------------------	-----------

<code>HAL_SMBUS_ERROR_HALTIMEOUT</code>	Timeout error
---	---------------

<code>HAL_SMBUS_ERROR_BUSTIMEOUT</code>	Bus Timeout error
---	-------------------

<code>HAL_SMBUS_ERROR_ALERT</code>	Alert error
------------------------------------	-------------

<code>HAL_SMBUS_ERROR_PECERR</code>	PEC error
-------------------------------------	-----------

**IO operation functions**

`HAL_SMBUS_EnableListen_IT`

`HAL_SMBUS_AddrCallback`

`HAL_SMBUS_ListenCpltCallback`

**SMBUS Exported Macros**

`_HAL_SMBUS_RESET_HANDLE_STATE` **Description:**

- Reset SMBUS handle state.

**Parameters:**

- `_HANDLE_`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.

**Return value:**

- None

`_HAL_SMBUS_ENABLE_IT`

**Description:**

- Enable or disable the specified SMBUS interrupts.

**Parameters:**

- `_HANDLE_`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- `_INTERRUPT_`: specifies the interrupt source to enable or disable. This parameter can be one of the

following values:

- SMBUS\_IT\_ERRI: Errors interrupt enable
- SMBUS\_IT\_TCI: Transfer complete interrupt enable
- SMBUS\_IT\_STOPI: STOP detection interrupt enable
- SMBUS\_IT\_NACKI: NACK received interrupt enable
- SMBUS\_IT\_ADDRI: Address match interrupt enable
- SMBUS\_IT\_RXI: RX interrupt enable
- SMBUS\_IT\_TXI: TX interrupt enable

**Return value:**

- None

`__HAL_SMBUS_DISABLE_IT`  
`__HAL_SMBUS_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- `__INTERRUPT__`: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS\_IT\_ERRI: Errors interrupt enable
  - SMBUS\_IT\_TCI: Transfer complete interrupt enable
  - SMBUS\_IT\_STOPI: STOP detection interrupt enable
  - SMBUS\_IT\_NACKI: NACK received interrupt enable
  - SMBUS\_IT\_ADDRI: Address match interrupt enable
  - SMBUS\_IT\_RXI: RX interrupt enable
  - SMBUS\_IT\_TXI: TX interrupt enable

**Return value:**

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

`SMBUS_FLAG_MASK`

**Description:**

- Checks whether the specified SMBUS

flag is set or not.

#### Parameters:

- \_\_HANDLE\_\_: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS\_FLAG\_TXE: Transmit data register empty
  - SMBUS\_FLAG\_RXIS: Transmit interrupt status
  - SMBUS\_FLAG\_RXNE: Receive data register not empty
  - SMBUS\_FLAG\_ADDR: Address matched (slave mode)
  - SMBUS\_FLAG\_AF\_NACK: received flag
  - SMBUS\_FLAG\_STOPF: STOP detection flag
  - SMBUS\_FLAG\_TC: Transfer complete (master mode)
  - SMBUS\_FLAG\_TCR: Transfer complete reload
  - SMBUS\_FLAG\_BERR: Bus error
  - SMBUS\_FLAG\_ARLO: Arbitration lost
  - SMBUS\_FLAG\_OVR: Overrun/Underrun
  - SMBUS\_FLAG\_PECERR: PEC error in reception
  - SMBUS\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
  - SMBUS\_FLAG\_ALERT: SMBus alert
  - SMBUS\_FLAG\_BUSY: Bus busy
  - SMBUS\_FLAG\_DIR: Transfer direction (slave mode)

#### Return value:

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

\_HAL\_SMBUS\_GET\_FLAG  
\_HAL\_SMBUS\_CLEAR\_FLAG

#### Description:

- Clears the SMBUS pending flags which are cleared by writing 1 in a specific bit.

#### Parameters:

- \_\_HANDLE\_\_: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the

- SMBUS peripheral.
- `__FLAG__`: specifies the flag to clear.  
This parameter can be any combination of the following values:
    - `SMBUS_FLAG_ADDR`: Address matched (slave mode)
    - `SMBUS_FLAG_AF`: NACK received flag
    - `SMBUS_FLAG_STOPF`: STOP detection flag
    - `SMBUS_FLAG_BERR`: Bus error
    - `SMBUS_FLAG_ARLO`: Arbitration lost
    - `SMBUS_FLAG_OVR`: Overrun/Underrun
    - `SMBUS_FLAG_PECERR`: PEC error in reception
    - `SMBUS_FLAG_TIMEOUT`: Timeout or Tlow detection flag
    - `SMBUS_FLAG_ALERT`: SMBus alert

**Return value:**

- None

`__HAL_SMBUS_ENABLE`  
`__HAL_SMBUS_DISABLE`  
`__SMBUS_RESET_CR1`  
`__SMBUS_RESET_CR2`  
`__SMBUS_GENERATE_START`  
`__SMBUS_GET_ADDR_MATCH`  
`__SMBUS_GET_DIR`  
`__SMBUS_GET_STOP_MODE`  
`__SMBUS_GET_PEC_MODE`  
`__SMBUS_GET_ALERT_ENABLE`  
`__HAL_SMBUS_GENERATE_NACK`  
`IS_SMBUS_OWN_ADDRESS1`  
`IS_SMBUS_OWN_ADDRESS2`

***SMBUS Flag Definition***

`SMBUS_FLAG_TXE`  
`SMBUS_FLAG_RXIS`  
`SMBUS_FLAG_RXNE`  
`SMBUS_FLAG_ADDR`  
`SMBUS_FLAG_AF`  
`SMBUS_FLAG_STOPF`

SMBUS\_FLAG\_TC  
SMBUS\_FLAG\_TCR  
SMBUS\_FLAG\_BERR  
SMBUS\_FLAG\_ARLO  
SMBUS\_FLAG\_OVR  
SMBUS\_FLAG\_PECERR  
SMBUS\_FLAG\_TIMEOUT  
SMBUS\_FLAG\_ALERT  
SMBUS\_FLAG\_BUSY  
SMBUS\_FLAG\_DIR

***SMBUS General Call Enabling***

SMBUS\_GENERALCALL\_DISABLE  
SMBUS\_GENERALCALL\_ENABLE  
IS\_SMBUS\_GENERAL\_CALL

***SMBUS Interrupt Configuration Definition***

SMBUS\_IT\_ERRI  
SMBUS\_IT\_TCI  
SMBUS\_IT\_STOPI  
SMBUS\_IT\_NACKI  
SMBUS\_IT\_ADDRI  
SMBUS\_IT\_RXI  
SMBUS\_IT\_TXI  
SMBUS\_IT\_TX  
SMBUS\_IT\_RX  
SMBUS\_IT\_ALERT  
SMBUS\_IT\_ADDR

***SMBUS Nostretch Enabling***

SMBUS\_NOSTRETCH\_DISABLE  
SMBUS\_NOSTRETCH\_ENABLE  
IS\_SMBUS\_NO\_STRETCH

***SMBUS Own Address2 Masks***

SMBUS\_OA2\_NOMASK  
SMBUS\_OA2\_MASK01  
SMBUS\_OA2\_MASK02  
SMBUS\_OA2\_MASK03  
SMBUS\_OA2\_MASK04

SMBUS\_OA2\_MASK05

SMBUS\_OA2\_MASK06

SMBUS\_OA2\_MASK07

IS\_SMBUS\_OWN\_ADDRESS2\_MASK

**SMBUS Packet Error Check Enabling**

SMBUS\_PEC\_DISABLE

SMBUS\_PEC\_ENABLE

IS\_SMBUS\_PEC

**SMBUS Peripheral Mode**

SMBUS\_PERIPHERAL\_MODE\_SMBUS\_HOST

SMBUS\_PERIPHERAL\_MODE\_SMBUS\_SLAVE

SMBUS\_PERIPHERAL\_MODE\_SMBUS\_SLAVE\_ARP

IS\_SMBUS\_PERIPHERAL\_MODE

**SMBUS Mode Definition**

SMBUS\_SOFTEND\_MODE

SMBUS\_RELOAD\_MODE

SMBUS\_AUTOEND\_MODE

SMBUS\_SENDPEC\_MODE

IS\_SMBUS\_TRANSFER\_MODE

**SMBUS StartStop Mode Definition**

SMBUS\_NO\_STARTSTOP

SMBUS\_GENERATE\_STOP

SMBUS\_GENERATE\_START\_READ

SMBUS\_GENERATE\_START\_WRITE

IS\_SMBUS\_TRANSFER\_REQUEST

**SMBUS State**

HAL_SMBUS_STATE_RESET	SMBUS not yet initialized or disabled
-----------------------	---------------------------------------

HAL_SMBUS_STATE_READY	SMBUS initialized and ready for use
-----------------------	-------------------------------------

HAL_SMBUS_STATE_BUSY	SMBUS internal process is ongoing
----------------------	-----------------------------------

HAL_SMBUS_STATE_MASTER_BUSY_TX	Master Data Transmission process is ongoing
--------------------------------	---

HAL_SMBUS_STATE_MASTER_BUSY_RX	Master Data Reception process is ongoing
--------------------------------	--

HAL_SMBUS_STATE_SLAVE_BUSY_TX	Slave Data Transmission process is ongoing
-------------------------------	--

HAL_SMBUS_STATE_SLAVE_BUSY_RX	Slave Data Reception process is ongoing
-------------------------------	---

HAL_SMBUS_STATE_TIMEOUT	Timeout state
-------------------------	---------------

HAL_SMBUS_STATE_ERROR	Reception process is ongoing
-----------------------	------------------------------

HAL_SMBUS_STATE_LISTEN	Address Listen Mode is ongoing
<b>SMBUS Transfer Request Definition</b>	
SMBUS_FIRST_FRAME	
SMBUS_NEXT_FRAME	
SMBUS_FIRST_AND_LAST_FRAME_NO_PEC	
SMBUS_LAST_FRAME_NO_PEC	
SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC	
SMBUS_LAST_FRAME_WITH_PEC	
IS_SMBUS_TRANSFER_OPTIONS_REQUEST	

## 44 HAL SPI Generic Driver

### 44.1 SPI Firmware driver registers structures

#### 44.1.1 SPI\_InitTypeDef

##### Data Fields

- *uint32\_t Mode*
- *uint32\_t Direction*
- *uint32\_t DataSize*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t NSS*
- *uint32\_t BaudRatePrescaler*
- *uint32\_t FirstBit*
- *uint32\_t TIMode*
- *uint32\_t CRCCalculation*
- *uint32\_t CRCPolynomial*

##### Field Documentation

- ***uint32\_t SPI\_InitTypeDef::Mode***  
Specifies the SPI operating mode. This parameter can be a value of [\*\*SPI\\_mode\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::Direction***  
Specifies the SPI Directional mode state. This parameter can be a value of [\*\*SPI\\_Direction\\_mode\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::DataSize***  
Specifies the SPI data size. This parameter can be a value of [\*\*SPI\\_data\\_size\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::CLKPolarity***  
Specifies the serial clock steady state. This parameter can be a value of [\*\*SPI\\_Clock\\_Polarity\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::CLKPhase***  
Specifies the clock active edge for the bit capture. This parameter can be a value of [\*\*SPI\\_Clock\\_Phase\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::NSS***  
Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [\*\*SPI\\_Slave\\_Select\\_management\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::BaudRatePrescaler***  
Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [\*\*SPI\\_BaudRate\\_Prescaler\*\*](#)  
**Note:** The communication clock is derived from the master clock. The slave clock does not need to be set
- ***uint32\_t SPI\_InitTypeDef::FirstBit***  
Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [\*\*SPI\\_MSB\\_LSB\\_transmission\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::TIMode***  
Specifies if the TI mode is enabled or not. This parameter can be a value of [\*\*SPI\\_TI\\_mode\*\*](#)

- ***uint32\_t SPI\_InitTypeDef::CRCCalculation***  
Specifies if the CRC calculation is enabled or not. This parameter can be a value of [\*\*SPI\\_CRC\\_Calculation\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::CRCPolynomial***  
Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min\_Data = 0 and Max\_Data = 65535

#### 44.1.2 [\\_\\_SPI\\_HandleTypeDef](#)

##### Data Fields

- ***SPI\_TypeDef \* Instance***
- ***SPI\_InitTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***void(\* RxISR***
- ***void(\* TxISR***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_SPI\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

##### Field Documentation

- ***SPI\_TypeDef\* \_\_SPI\_HandleTypeDef::Instance***  
SPI 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
- ***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
- ***uint16\_t \_\_SPI\_HandleTypeDef::RxXferCount***  
SPI Rx Transfer Counter
- ***DMA\_HandleTypeDef\* \_\_SPI\_HandleTypeDef::hdmatx***  
SPI Tx DMA handle parameters
- ***DMA\_HandleTypeDef\* \_\_SPI\_HandleTypeDef::hdmarx***  
SPI Rx DMA handle parameters
- ***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
- **HAL\_LockTypeDef \_\_SPI\_HandleTypeDef::Lock**  
SPI locking object
- **\_IO HAL\_SPI\_StateTypeDef \_\_SPI\_HandleTypeDef::State**  
SPI communication state
- **\_IO uint32\_t \_\_SPI\_HandleTypeDef::ErrorCode**  
SPI Error code

## 44.2 SPI Firmware driver API description

### 44.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI\_HandleTypeDef handle structure, for example: SPI\_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL\_SPI\_MspInit ()API:
  - a. Enable the SPIx interface clock
  - b. SPI pins configuration
    - Enable the clock for the SPI GPIOs
    - Configure these SPI pins as alternate function push-pull
  - c. NVIC configuration if you need to use interrupt process
    - Configure the SPIx interrupt priority
    - Enable the NVIC SPI IRQ handle
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive Channel
    - Enable the DMAx clock
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx Channel
    - Associate the initialized hdma\_tx(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, Direction , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL\_SPI\_Init() API:
  - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc by calling the customized HAL\_SPI\_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
  - a. Master 2Lines RxOnly
  - b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL\_SPI\_DMAPause()/ HAL\_SPI\_DMAStop() only under the SPI callbacks

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI modes (refer to the product line reference manual for the maximum SPI frequency vs data).

#### 44.2.2 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\(\)\*\*\*](#)

#### 44.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SPI\_TxCpltCallback(), HAL\_SPI\_RxCpltCallback() and HAL\_SPI\_TxRx\_CpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL\_SPI\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
  - HAL\_SPI\_Transmit() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_Receive() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_TransmitReceive() in full duplex mode
3. Non Blocking mode API's with Interrupt are :
  - HAL\_SPI\_Transmit\_IT() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_Receive\_IT() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_TransmitReceive\_IT() in full duplex mode
  - HAL\_SPI\_IRQHandler()
4. Non Blocking mode functions with DMA are :
  - HAL\_SPI\_Transmit\_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_Receive\_DMA() in 1Line (simplex) and 2Lines (full duplex) mode

- HAL\_SPI\_TransmitReceive\_DMA() in full duplex mode
- 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_SPI\_TxCpltCallback()
  - HAL\_SPI\_RxCpltCallback()
  - HAL\_SPI\_TxRxCpltCallback()
  - HAL\_SPI\_TxHalfCpltCallback()
  - HAL\_SPI\_RxHalfCpltCallback()
  - HAL\_SPI\_TxRxHalfCpltCallback()
  - HAL\_SPI\_ErrorCallback()

This section contains the following APIs:

- [\*HAL\\_SPI\\_Transmit\(\)\*](#)
- [\*HAL\\_SPI\\_Receive\(\)\*](#)
- [\*HAL\\_SPI\\_TransmitReceive\(\)\*](#)
- [\*HAL\\_SPI\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_SPI\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_SPI\\_TransmitReceive\\_IT\(\)\*](#)
- [\*HAL\\_SPI\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_SPI\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_SPI\\_TransmitReceive\\_DMA\(\)\*](#)
- [\*HAL\\_SPI\\_DMAPause\(\)\*](#)
- [\*HAL\\_SPI\\_DMAResume\(\)\*](#)
- [\*HAL\\_SPI\\_DMAStop\(\)\*](#)
- [\*HAL\\_SPI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SPI\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_TxRxCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_TxRxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_ErrorCallback\(\)\*](#)

#### 44.2.4 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\(\)\*](#)

#### 44.2.5 Detailed description of functions

##### HAL\_SPI\_Init

Function Name	<a href="#"><b>HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)</b></a>
Function Description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>

Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_SPI_DelInit</b>	
Function Name	<b>HAL_StatusTypeDef HAL_SPI_DelInit (SPI_HandleTypeDef * hspi)</b>
Function Description	DeInitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_SPI_MspInit</b>	
Function Name	<b>void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
<b>HAL_SPI_MspDelInit</b>	
Function Name	<b>void HAL_SPI_MspDelInit (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
<b>HAL_SPI_Transmit</b>	
Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be sent</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_SPI_Receive</b>	
Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>

- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration
  
- **Return values**
- **HAL:** status

### **HAL\_SPI\_TransmitReceive**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pTxData:</b> pointer to transmission data buffer</li> <li>• <b>pRxData:</b> pointer to reception data buffer to be</li> <li>• <b>Size:</b> amount of data to be sent</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_SPI\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_SPI\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_SPI\_TransmitReceive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Transmit and Receive an amount of data in no-blocking mode with

Interrupt.

Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li><b>pTxData:</b> pointer to transmission data buffer</li> <li><b>pRxData:</b> pointer to reception data buffer to be</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_SPI\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_DMA(SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_SPI\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_DMA(SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the CRC feature is enabled the pData Length must be Size + 1.</li> </ul>

### HAL\_SPI\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA(SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Transmit and Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li><b>pTxData:</b> pointer to transmission data buffer</li> <li><b>pRxData:</b> pointer to reception data buffer</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the CRC feature is enabled the pRxData Length must be Size + 1</li> </ul>

**HAL\_SPI\_DMAPause**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DM_PAUSE (SPI_HandleTypeDef * hspi)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SPI\_DMAResume**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SPI\_DMAStop**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SPI\_IRQHandler**

Function Name	<b>void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)</b>
Function Description	This function handles SPI interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_SPI\_TxCpltCallback**

Function Name	<b>void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_SPI\_RxCpltCallback**

Function Name	<b>void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)</b>
---------------	---

---

Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_SPI\_TxRxCpltCallback**

Function Name	<b>void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_SPI\_ErrorCallback**

Function Name	<b>void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_SPI\_TxHalfCpltCallback**

Function Name	<b>void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_SPI\_RxHalfCpltCallback**

Function Name	<b>void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **HAL\_SPI\_TxRxHalfCpltCallback**

Function Name	<b>void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>

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

### **HAL\_SPI\_GetState**

Function Name	<b>HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)</b>
Function Description	Return the SPI state.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>SPI:</b> state</li></ul>

### **HAL\_SPI\_GetError**

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

## **44.3 SPI Firmware driver defines**

### **44.3.1 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 SPIx where x: 1 or 2 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 SPIx where x: 1 or 2 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 SPIx where x: 1 or 2 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

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

**Parameters:**

- HANDLE: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 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

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

**Parameters:**

- HANDLE: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.
- FLAG: specifies the flag to check. This parameter can be one of the following values:
  - SPI\_FLAG\_RXNE: Receive buffer not empty flag
  - SPI\_FLAG\_TXE: Transmit buffer empty flag
  - SPI\_FLAG\_CRCERR: CRC error flag

- SPI\_FLAG\_MODF: Mode fault flag
- SPI\_FLAG\_OVR: Overrun flag
- SPI\_FLAG\_BSY: Busy flag
- SPI\_FLAG\_FRE: Frame format error flag

**Return value:**

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

[\\_\\_HAL\\_SPI\\_CLEAR\\_CRCERRFLAG](#)**Description:**

- Clear the SPI CRCERR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 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 SPIx where x: 1 or 2 to select the SPI peripheral.

**Return value:**

- None

[\\_\\_HAL\\_SPI\\_CLEAR\\_OVRFAG](#)**Description:**

- Clear the SPI OVR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

**Return value:**

- None

[\\_\\_HAL\\_SPI\\_CLEAR\\_FREFLAG](#)**Description:**

- Clear the SPI FRE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

**Return value:**

- None

\_\_HAL\_SPI\_ENABLE**Description:**

- Enables the SPI.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI Handle.  
This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

**Return value:**

- None

\_\_HAL\_SPI\_DISABLE**Description:**

- Disables the SPI.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI Handle.  
This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

**Return value:**

- None

***SPI Flag definition***

SPI\_FLAG\_RXNE

SPI\_FLAG\_TXE

SPI\_FLAG\_CRCERR

SPI\_FLAG\_MODF

SPI\_FLAG\_OVR

SPI\_FLAG\_BSY

SPI\_FLAG\_FRE

***SPI Interrupt configuration definition***

SPI\_IT\_TXE

SPI\_IT\_RXNE

SPI\_IT\_ERR

***SPI mode***

SPI\_MODE\_SLAVE

SPI\_MODE\_MASTER

***SPI MSB LSB transmission***

SPI\_FIRSTBIT\_MSB

SPI\_FIRSTBIT\_LSB

***SPI Slave Select management***

SPI\_NSS\_SOFT

SPI\_NSS\_HARD\_INPUT

SPI\_NSS\_HARD\_OUTPUT

***SPI TI mode***

SPI\_TIMODE\_DISABLE

SPI\_TIMODE\_ENABLE

## 45 HAL TIM Generic Driver

### 45.1 TIM Firmware driver registers structures

#### 45.1.1 TIM\_Base\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t CounterMode*
- *uint32\_t Period*
- *uint32\_t ClockDivision*

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

#### 45.1.2 TIM\_OC\_InitTypeDef

##### Data Fields

- *uint32\_t OCMode*
- *uint32\_t Pulse*
- *uint32\_t OCPolarity*
- *uint32\_t OCFastMode*

##### 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::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.

### 45.1.3 TIM\_OnePulse\_InitTypeDef

#### Data Fields

- ***uint32\_t OCMode***
- ***uint32\_t Pulse***
- ***uint32\_t OCPolarity***
- ***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::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

### 45.1.4 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

#### 45.1.5 **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

#### 45.1.6 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

#### 45.1.7 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

### 45.1.8 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

### 45.1.9 TIM\_HandleTypeDef

#### Data Fields

- *TIM\_TypeDef \* Instance*
- *TIM\_Base\_InitTypeDef Init*
- *HAL\_TIM\_ActiveChannel Channel*
- *DMA\_HandleTypeDef \* hdma*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_TIM\_StateTypeDef State*

#### Field Documentation

- *TIM\_TypeDef\* TIM\_HandleTypeDef::Instance*  
Register base address
- *TIM\_Base\_InitTypeDef TIM\_HandleTypeDef::Init*  
TIM Time Base required parameters
- *HAL\_TIM\_ActiveChannel TIM\_HandleTypeDef::Channel*  
Active channel
- *DMA\_HandleTypeDef\* TIM\_HandleTypeDef::hdma[7]*  
DMA Handlers array This array is accessed by a [DMA\\_Handle\\_index](#)
- *HAL\_LockTypeDef TIM\_HandleTypeDef::Lock*  
Locking object

- `_IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State`  
TIM operation state

## 45.2 TIM Firmware driver API description

### 45.2.1 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
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

### 45.2.2 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: `HAL_TIM_Base_Start()`, `HAL_TIM_Base_Start_DMA()`, `HAL_TIM_Base_Start_IT()`, `HAL_TIM_OC_Start()`, `HAL_TIM_OC_Start_DMA()`, `HAL_TIM_OC_Start_IT()`, `HAL_TIM_IC_Start()`, `HAL_TIM_IC_Start_DMA()`, `HAL_TIM_IC_Start_IT()`, `HAL_TIM_PWM_Start()`, `HAL_TIM_PWM_Start_DMA()`, `HAL_TIM_PWM_Start_IT()`, `HAL_TIM_OnePulse_Start()`, `HAL_TIM_OnePulse_Start_IT()`, `HAL_TIM_Encoder_Start()`, `HAL_TIM_Encoder_Start_DMA()` or `HAL_TIM_Encoder_Start_IT()`
6. The DMA Burst is managed with the two following functions:  
`HAL_TIM_DMABurst_WriteStart` `HAL_TIM_DMABurst_ReadStart`

### 45.2.3 Timer Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Timer Base.
- Stop the Timer Base.
- Start the Timer Base and enable interrupt.
- Stop the Timer Base and disable interrupt.
- Start the Timer Base and enable DMA transfer.
- Stop the Timer 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()`

### 45.2.4 Peripheral State functions

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

This section contains the following APIs:

- `HAL_TIM_Base_GetState()`
- `HAL_TIM_OC_GetState()`
- `HAL_TIM_PWM_GetState()`
- `HAL_TIM_IC_GetState()`
- `HAL_TIM_OnePulse_GetState()`
- `HAL_TIM_Encoder_GetState()`
- `TIM_DMAError()`
- `TIM_DMADelayPulseCplt()`
- `TIM_DMACaptureCplt()`

## 45.2.5 Detailed description of functions

### HAL\_TIM\_Base\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_Base\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_Base\_MspInit

Function Name	<b>void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_TIM\_Base\_MspDeInit

Function Name	<b>void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Base MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_TIM\_Base\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)</b>
Function Description	Starts the TIM Base generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_Base\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)</b>
---------------	---

Function Description	Stops the TIM Base generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_TIM\_Base\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)</b>
Function Description	Starts the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_TIM\_Base\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_TIM\_Base\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>pData:</b> The source Buffer address.</li> <li>• <b>Length:</b> The length of data to be transferred from memory to peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_TIM\_Base\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_TIM\_OC\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

---

Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> TIM Output Compare handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_TIM\_OC\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_DelInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> TIM Output Compare handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_TIM\_OC\_MspInit

Function Name	<b>void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_TIM\_OC\_MspDelInit

Function Name	<b>void HAL_TIM_OC_MspDelInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_TIM\_OC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM handle</li><li>• <b>Channel:</b> TIM Channel to be enabled. This parameter can be one of the following values:<ul style="list-style-type: none"><li>– TIM_CHANNEL_1: TIM Channel 1 selected</li><li>– TIM_CHANNEL_2: TIM Channel 2 selected</li><li>– TIM_CHANNEL_3: TIM Channel 3 selected</li><li>– TIM_CHANNEL_4: TIM Channel 4 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_TIM\_OC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM handle</li><li>• <b>Channel:</b> TIM Channel to be disabled. This parameter can</li></ul>

be one of the following values:

- TIM\_CHANNEL\_1: TIM Channel 1 selected
- TIM\_CHANNEL\_2: TIM Channel 2 selected
- TIM\_CHANNEL\_3: TIM Channel 3 selected
- TIM\_CHANNEL\_4: TIM Channel 4 selected

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

### HAL\_TIM\_OC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channel to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channel to be disabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channel to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> </ul> </li> </ul>

- **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**

- |                             |   |
|-----------------------------|---|
| <b>Function Name</b>        | <b>HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA<br/>(TIM_HandleTypeDef * htim, uint32_t Channel)</b>   |
| <b>Function Description</b> | Stops the TIM Output Compare signal generation in DMA mode.   |
| <b>Parameters</b>           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channel to be disabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1:</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2:</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3:</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4:</b> TIM Channel 4 selected</li> </ul> </li> </ul> |
| <b>Return values</b>        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>  |

### **HAL\_TIM\_PWM\_Init**

- |                             |  |
|-----------------------------|--|
| <b>Function Name</b>        | <b>HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)</b>   |
| <b>Function Description</b> | Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle. |
| <b>Parameters</b>           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>  |
| <b>Return values</b>        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>   |

### **HAL\_TIM\_PWM\_DeInit**

- |                             |   |
|-----------------------------|---|
| <b>Function Name</b>        | <b>HAL_StatusTypeDef HAL_TIM_PWM_DeInit<br/>(TIM_HandleTypeDef * htim)</b>    |
| <b>Function Description</b> | Deinitializes the TIM peripheral.   |
| <b>Parameters</b>           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul> |
| <b>Return values</b>        | <ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>        |

### **HAL\_TIM\_PWM\_MspInit**

- |                             |   |
|-----------------------------|---|
| <b>Function Name</b>        | <b>void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)</b>                    |
| <b>Function Description</b> | Initializes the TIM PWM MSP.  |
| <b>Parameters</b>           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul> |
| <b>Return values</b>        | <ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>              |

**HAL\_TIM\_PWM\_MspDeInit**

Function Name	<b>void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_PWM\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_PWM\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_PWM\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_PWM_Stop_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_PWM_Start_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> <li><b>pData:</b> The source Buffer address. This buffer contains the values which will be loaded inside the capture/compare registers.</li> <li><b>Length:</b> The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_PWM_Stop_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>

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

**HAL\_TIM\_IC\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> TIM Input Capture handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

**HAL\_TIM\_IC\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> TIM Input Capture handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

**HAL\_TIM\_IC\_MspInit**

Function Name	<b>void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM INput Capture MSP.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

**HAL\_TIM\_IC\_MspDeInit**

Function Name	<b>void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	Deinitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

**HAL\_TIM\_IC\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
<b>HAL_TIM_IC_Stop</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
<b>HAL_TIM_IC_Start_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
<b>HAL_TIM_IC_Stop_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
<b>HAL_TIM_IC_Start_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * </b>

**pData, uint16\_t Length)**

Function Description	Starts the TIM Input Capture measurement on in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> <li>• <b>pData:</b> The destination Buffer address.</li> <li>• <b>Length:</b> The length of data to be transferred from TIM peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_IC\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA</b> <b>(TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Input Capture measurement on in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_OnePulse\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Init</b> <b>(TIM_HandleTypeDef * htim, uint32_t OnePulseMode)</b>
Function Description	Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM OnePulse handle</li> <li>• <b>OnePulseMode:</b> Select the One pulse mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_OPmode_SINGLE: Only one pulse will be generated.</li> <li>- TIM_OPmode_REPETITIVE: Repetitive pulses will be generated.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_OnePulse\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit</b> <b>(TIM_HandleTypeDef * htim)</b>
---------------	---

Function Description	Deinitializes the TIM One Pulse.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM One Pulse handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OnePulse\_MspInit

Function Name	<b>void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_TIM\_OnePulse\_MspDeInit

Function Name	<b>void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	Deinitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_TIM\_OnePulse\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>OutputChannel:</b> : TIM Channels to be enabled. This parameter is not used since both channels TIM_CHANNEL_1 and TIM_CHANNEL_2 are automatically selected.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OnePulse\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>OutputChannel:</b> : TIM Channels to be disable. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OnePulse\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
---------------	---

Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>OutputChannel:</b> TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OnePulse\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT</b> <b>(TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>OutputChannel:</b> TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_Encoder\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Init</b> <b>(TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)</b>
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM Encoder Interface handle</li> <li>• <b>sConfig:</b> TIM Encoder Interface configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_Encoder\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_DelInit</b> <b>(TIM_HandleTypeDef * htim)</b>
Function Description	Deinitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM Encoder handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_Encoder\_MspInit

Function Name	<b>void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**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 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 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 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	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_Encoder_Stop_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_Encoder_Start_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)</b>
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_ALL : TIM Channel 1 and 2 selected</li> </ul> </li> <li><b>pData1:</b> The destination Buffer address for IC1.</li> <li><b>pData2:</b> The destination Buffer address for IC2.</li> <li><b>Length:</b> The length of data to be transferred from TIM peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_Encoder_Stop_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

**HAL\_TIM\_IRQHandler**

Function Name	<b>void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)</b>
Function Description	This function handles TIM interrupts requests.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_OC\_ConfigChannel**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>sConfig:</b> TIM Output Compare configuration structure</li> <li>• <b>Channel:</b> TIM Channel to be configure. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_PWM\_ConfigChannel**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>sConfig:</b> TIM PWM configuration structure</li> <li>• <b>Channel:</b> TIM Channel to be configured. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_IC\_ConfigChannel**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>sConfig:</b> TIM Input Capture configuration structure</li> <li><b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_TIM\_OnePulse\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)</b>
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>sConfig:</b> TIM One Pulse configuration structure</li> <li><b>OutputChannel:</b> TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul> </li> <li><b>InputChannel:</b> TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_TIM\_ConfigOCrefClear

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)</b>
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>sClearInputConfig:</b> pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.</li> <li><b>Channel:</b> specifies the TIM Channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

**HAL\_TIM\_ConfigClockSource**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)</b>
Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>sClockSourceConfig:</b> pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_ConfigTI1Input**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)</b>
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>TI1_Selection:</b> Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>TIM_TI1SELECTION_CH1:</b> The TIMx_CH1 pin is connected to TI1 input</li> <li>– <b>TIM_TI1SELECTION_XORCOMBINATION:</b> The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_SlaveConfigSynchronization**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)</b>
Function Description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>sSlaveConfig:</b> pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the ) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_TIM\_SlaveConfigSynchronization\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)</b>
---------------	--

Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle.</li> <li><b>sSlaveConfig:</b> pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_TIM\_DMABurst\_WriteStart

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)</b>
Function Description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>BurstBaseAddress:</b> TIM Base address from when the DMA will starts the Data write. This parameters can be one of the following values: <ul style="list-style-type: none"> <li>TIM_DMABASE_CR1</li> <li>TIM_DMABASE_CR2</li> <li>TIM_DMABASE_SMCR</li> <li>TIM_DMABASE_DIER</li> <li>TIM_DMABASE_SR</li> <li>TIM_DMABASE_EGR</li> <li>TIM_DMABASE_CCMR1</li> <li>TIM_DMABASE_CCMR2</li> <li>TIM_DMABASE_CCER</li> <li>TIM_DMABASE_CNT</li> <li>TIM_DMABASE_PSC</li> <li>TIM_DMABASE_ARR</li> <li>TIM_DMABASE_CCR1</li> <li>TIM_DMABASE_CCR2</li> <li>TIM_DMABASE_CCR3</li> <li>TIM_DMABASE_CCR4</li> <li>TIM_DMABASE_DCR</li> </ul> </li> <li><b>BurstRequestSrc:</b> TIM DMA Request sources. This parameters can be one of the following values: <ul style="list-style-type: none"> <li>TIM_DMA_UPDATE: TIM update Interrupt source</li> <li>TIM_DMA_CC1: TIM Capture Compare 1 DMA source</li> <li>TIM_DMA_CC2: TIM Capture Compare 2 DMA source</li> <li>TIM_DMA_CC3: TIM Capture Compare 3 DMA source</li> <li>TIM_DMA_CC4: TIM Capture Compare 4 DMA source</li> <li>TIM_DMA_TRIGGER: TIM Trigger DMA source</li> </ul> </li> <li><b>BurstBuffer:</b> The Buffer address.</li> <li><b>BurstLength:</b> DMA Burst length. This parameter can be one value between TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS .</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TIM_DMABurst_WriteStop</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)</b>
Function Description	Stops the TIM DMA Burst mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>BurstRequestSrc:</b> TIM DMA Request sources to disable</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)</b>
Function Description	Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> <li><b>BurstBaseAddress:</b> TIM Base address from when the DMA will starts the Data read. This parameters can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_DMABASE_CR1</li> <li>- TIM_DMABASE_CR2</li> <li>- TIM_DMABASE_SMCR</li> <li>- TIM_DMABASE_DIER</li> <li>- TIM_DMABASE_SR</li> <li>- TIM_DMABASE_EGR</li> <li>- TIM_DMABASE_CCMR1</li> <li>- TIM_DMABASE_CCMR2</li> <li>- TIM_DMABASE_CCER</li> <li>- TIM_DMABASE_CNT</li> <li>- TIM_DMABASE_PSC</li> <li>- TIM_DMABASE_ARR</li> <li>- TIM_DMABASE_CCR1</li> <li>- TIM_DMABASE_CCR2</li> <li>- TIM_DMABASE_CCR3</li> <li>- TIM_DMABASE_CCR4</li> <li>- TIM_DMABASE_DCR</li> </ul> </li> <li><b>BurstRequestSrc:</b> TIM DMA Request sources. This parameters can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_DMA_UPDATE: TIM update Interrupt source</li> <li>- TIM_DMA_CC1: TIM Capture Compare 1 DMA source</li> <li>- TIM_DMA_CC2: TIM Capture Compare 2 DMA source</li> <li>- TIM_DMA_CC3: TIM Capture Compare 3 DMA source</li> <li>- TIM_DMA_CC4: TIM Capture Compare 4 DMA source</li> <li>- TIM_DMA_TRIGGER: TIM Trigger DMA source</li> </ul> </li> <li><b>BurstBuffer:</b> The Buffer address.</li> <li><b>BurstLength:</b> DMA Burst length. This parameter can be one value between TIM_DMABURSTLENGTH_1TRANSFER and</li> </ul>

TIM\_DMABURSTLENGTH\_18TRANSFERS .

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

### HAL\_TIM\_DMABurst\_ReadStop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)</b>
Function Description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>BurstRequestSrc:</b> TIM DMA Request sources to disable.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TIM\_GenerateEvent

Function Name	<b>HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)</b>
Function Description	Generate a software event.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>EventSource:</b> specifies the event source. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_EventSource_Update: Timer update Event source</li> <li>- TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source</li> <li>- TIM_EventSource_CC2: Timer Capture Compare 2 Event source</li> <li>- TIM_EventSource_CC3: Timer Capture Compare 3 Event source</li> <li>- TIM_EventSource_CC4: Timer Capture Compare 4 Event source</li> <li>- TIM_EVENTSOURCE_TRIGGER : Timer Trigger Event source</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• TIM6 can only generate an update event.</li> </ul>

### HAL\_TIM\_ReadCapturedValue

Function Name	<b>uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- TIM_CHANNEL_1: TIM Channel 1 selected</li> <li>- TIM_CHANNEL_2: TIM Channel 2 selected</li> <li>- TIM_CHANNEL_3: TIM Channel 3 selected</li> <li>- TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Captured:</b> value</li> </ul>

**HAL\_TIM\_PeriodElapsedCallback**

Function Name	<b>void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_OC\_DelayElapsedCallback**

Function Name	<b>void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Output Compare callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_IC\_CaptureCallback**

Function Name	<b>void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Input Capture callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM IC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_PWM\_PulseFinishedCallback**

Function Name	<b>void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)</b>
Function Description	PWM Pulse finished callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_TriggerCallback**

Function Name	<b>void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_TIM\_ErrorCallback**

Function Name	<b>void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>

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 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 Output 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 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	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

**TIM\_DMADelayPulseCplt**

Function Name	<b>void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)</b>
Function Description	TIM DMA Delay Pulse complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**TIM\_DMAError**

Function Name	<b>void TIM_DMAError (DMA_HandleTypeDef * hdma)</b>
Function Description	TIM DMA error callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</b> pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**TIM\_DMACaptureCplt**

Function Name	<b>void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)</b>
Function Description	TIM DMA Capture complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 45.3 TIM Firmware driver defines

### 45.3.1 TIM

***TIM channels***

TIM\_CHANNEL\_1

TIM\_CHANNEL\_2

TIM\_CHANNEL\_3

TIM\_CHANNEL\_4

TIM\_CHANNEL\_ALL

***Clear input polarity***

TIM\_CLEARINPUTPOLARITY\_INVERTED      Polarity for ETRx pin

`TIM_CLEARINPUTPOLARITY_NONINVERTED` Polarity for ETRx pin

***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.

***Clear input source***

`TIM_CLEARINPUTSOURCE_ETR`

`TIM_CLEARINPUTSOURCE_NONE`

***Clock division***

`TIM_CLOCKDIVISION_DIV1`

`TIM_CLOCKDIVISION_DIV2`

`TIM_CLOCKDIVISION_DIV4`

***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

***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.

***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

**Counter mode**

TIM\_COUNTERMODE\_UP  
TIM\_COUNTERMODE\_DOWN  
TIM\_COUNTERMODE\_CENTERALIGNED1  
TIM\_COUNTERMODE\_CENTERALIGNED2  
TIM\_COUNTERMODE\_CENTERALIGNED3

**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\_CCR1  
TIM\_DMABASE\_CCR2  
TIM\_DMABASE\_CCR3  
TIM\_DMABASE\_CCR4  
TIM\_DMABASE\_DCR  
TIM\_DMABASE\_OR

**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

**DMA sources**

TIM\_DMA\_UPDATE  
TIM\_DMA\_CC1  
TIM\_DMA\_CC2  
TIM\_DMA\_CC3  
TIM\_DMA\_CC4  
TIM\_DMA\_TRIGGER

**Encoder\_Mode**

TIM\_ENCODERMODE\_TI1  
TIM\_ENCODERMODE\_TI2  
TIM\_ENCODERMODE\_TI12

**ETR polarity**

TIM\_ETRPOLARITY\_INVERTED      Polarity for ETR source

TIM\_ETRPOLARITY\_NONINVERTED      Polarity for ETR source

**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

**Event sources**

TIM\_EVENTSOURCE\_UPDATE  
TIM\_EVENTSOURCE\_CC1  
TIM\_EVENTSOURCE\_CC2  
TIM\_EVENTSOURCE\_CC3  
TIM\_EVENTSOURCE\_CC4

TIM\_EVENTSOURCE\_TRIGGER  
***TIM Exported Constants***  
IS\_TIM\_PERIOD  
IS\_TIM\_PRESCALER  
IS\_TIM\_COUNTER\_MODE  
IS\_TIM\_CLOCKDIVISION\_DIV  
IS\_TIM\_PWM\_MODE  
IS\_TIM\_OC\_MODE  
IS\_TIM\_FAST\_STATE  
IS\_TIM\_OC\_POLARITY  
IS\_TIM\_CHANNELS  
IS\_TIM\_OPM\_CHANNELS  
IS\_TIM\_IC\_POLARITY  
IS\_TIM\_IC\_PRESCALER  
IS\_TIM\_OPM\_MODE  
IS\_TIM\_ENCODER\_MODE  
IS\_TIM\_DMA\_SOURCE  
IS\_TIM\_EVENT\_SOURCE  
IS\_TIM\_CLOCKSOURCE  
IS\_TIM\_CLOCKPOLARITY  
IS\_TIM\_CLOCKPRESCALER  
IS\_TIM\_CLOCKFILTER  
IS\_TIM\_CLEARINPUT\_SOURCE  
IS\_TIM\_CLEARINPUT\_POLARITY  
IS\_TIM\_CLEARINPUT\_PRESCALER  
IS\_TIM\_CLEARINPUT\_FILTER  
IS\_TIM\_TRGO\_SOURCE  
IS\_TIM\_SLAVE\_MODE  
IS\_TIM\_MSM\_STATE  
IS\_TIM\_TRIGGER\_SELECTION  
IS\_TIM\_INTERNAL\_TRIGGEREVENT\_SELECTION  
IS\_TIM\_TRIGGERPOLARITY  
IS\_TIM\_TRIGGERPRESCALER  
IS\_TIM\_TRIGGERFILTER  
IS\_TIM\_TI1SELECTION  
IS\_TIM\_DMA\_BASE

IS\_TIM\_DMA\_LENGTH

IS\_TIM\_IC\_FILTER

**TIM Exported Macro**

`_HAL_TIM_RESET_HANDLE_ST  
ATE`

**Description:**

- Reset UART handle state.

**Parameters:**

- `_HANDLE_`: TIM handle

**Return value:**

- None

`_HAL_TIM_ENABLE`

**Description:**

- Enable the TIM peripheral.

**Parameters:**

- `_HANDLE_`: TIM handle

**Return value:**

- None

`TIM_CCER_CCxE_MASK`

`_HAL_TIM_DISABLE`

**Description:**

- Disable the TIM peripheral.

**Parameters:**

- `_HANDLE_`: TIM handle

**Return value:**

- None

`_HAL_TIM_ENABLE_IT`

`_HAL_TIM_ENABLE_DMA`

`_HAL_TIM_DISABLE_IT`

`_HAL_TIM_DISABLE_DMA`

`_HAL_TIM_GET_FLAG`

`_HAL_TIM_CLEAR_FLAG`

`_HAL_TIM_GET_IT_SOURCE`

`_HAL_TIM_CLEAR_IT`

`_HAL_TIM_IS_TIM_COUNTING_  
DOWN`

`_HAL_TIM_SET_PRESCALER`

`TIM_SET_ICPRESCALERVALUE`

`TIM_RESET_ICPRESCALERVAL  
UE`

`TIM_SET_CAPTUREPOLARITY`

TIM\_RESET\_CAPTUREPOLARITY  
Y

\_\_HAL\_TIM\_SET\_COMPARE

**Description:**

- Sets the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

**Parameters:**

- \_\_HANDLE\_\_: : TIM handle.
- \_\_CHANNEL\_\_: : TIM Channels to be configured. This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: TIM Channel 1 selected
  - TIM\_CHANNEL\_2: TIM Channel 2 selected
  - TIM\_CHANNEL\_3: TIM Channel 3 selected
  - TIM\_CHANNEL\_4: TIM Channel 4 selected
- \_\_COMPARE\_\_: specifies the Capture Compare register new value.

**Return value:**

- None

\_\_HAL\_TIM\_GET\_COMPARE

**Description:**

- Gets the TIM Capture Compare Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: : TIM handle.
- \_\_CHANNEL\_\_: : TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: get capture/compare 1 register value
  - TIM\_CHANNEL\_2: get capture/compare 2 register value
  - TIM\_CHANNEL\_3: get capture/compare 3 register value
  - TIM\_CHANNEL\_4: get capture/compare 4 register value

**Return value:**

- None

\_\_HAL\_TIM\_SET\_COUNTER

**Description:**

- Sets the TIM Counter Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: : TIM handle.

- COUNTER: specifies the Counter register new value.

**Return value:**

- None

\_HAL\_TIM\_GET\_COUNTER**Description:**

- Gets the TIM Counter Register value on runtime.

**Parameters:**

- HANDLE: TIM handle.

**Return value:**

- None

\_HAL\_TIM\_SET\_AUTORELOAD**Description:**

- Sets the TIM Autoreload Register value on runtime without calling another time any Init function.

**Parameters:**

- HANDLE: TIM handle.
- AUTORELOAD: specifies the Counter register new value.

**Return value:**

- None

\_HAL\_TIM\_GET\_AUTORELOAD**Description:**

- Gets the TIM Autoreload Register value on runtime.

**Parameters:**

- HANDLE: TIM handle.

**Return value:**

- None

\_HAL\_TIM\_SET\_CLOCKDIVISION**Description:**

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

**Parameters:**

- HANDLE: TIM handle.
- CKD: specifies the clock division value. This parameter can be one of the following value:
  - TIM\_CLOCKDIVISION\_DIV1
  - TIM\_CLOCKDIVISION\_DIV2
  - TIM\_CLOCKDIVISION\_DIV4

**Return value:**

- None

`__HAL_TIM_GET_CLOCKDIVISION`  
N

**Description:**

- Gets the TIM Clock Division value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None

`__HAL_TIM_SET_ICPRESCALER`

**Description:**

- Sets the TIM Input Capture prescaler on runtime without calling another time

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - `TIM_ICPSC_DIV1`: no prescaler
  - `TIM_ICPSC_DIV2`: capture is done once every 2 events
  - `TIM_ICPSC_DIV4`: capture is done once every 4 events
  - `TIM_ICPSC_DIV8`: capture is done once every 8 events

**Return value:**

- None

`__HAL_TIM_GET_ICPRESCALER`

**Description:**

- Gets the TIM Input Capture prescaler on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: get input capture 1 prescaler value
  - `TIM_CHANNEL_2`: get input capture 2

- prescaler value
- TIM\_CHANNEL\_3: get input capture 3 prescaler value
- TIM\_CHANNEL\_4: get input capture 4 prescaler value

**Return value:**

- None

**\_HAL\_TIM\_URS\_ENABLE****Description:**

- Set the Update Request Source (URS) bit of the TIMx\_CR1 register.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

**Return value:**

- None

**Notes:**

- When the URS 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 URS 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.

***Flag definition***

TIM\_FLAG\_UPDATE  
 TIM\_FLAG\_CC1  
 TIM\_FLAG\_CC2  
 TIM\_FLAG\_CC3  
 TIM\_FLAG\_CC4  
 TIM\_FLAG\_TRIGGER  
 TIM\_FLAG\_CC1OF  
 TIM\_FLAG\_CC2OF  
 TIM\_FLAG\_CC3OF  
 TIM\_FLAG\_CC4OF

***Input capture polarity***

TIM\_ICPOLARITY\_RISING  
 TIM\_ICPOLARITY\_FALLING  
 TIM\_ICPOLARITY\_BOTHEDGE

***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

***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
<b>IS_TIM_IC_SELECTION</b>	
<b><i>Input channel polarity</i></b>	
TIM_INPUTCHANNELPOLARITY_RISING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for TIx source
<b><i>Interrupt definition</i></b>	
TIM_IT_UPDATE	
TIM_IT_CC1	
TIM_IT_CC2	
TIM_IT_CC3	
TIM_IT_CC4	
TIM_IT_TRIGGER	
<b><i>TIM Master Mode Selection</i></b>	
TIM_TRGO_RESET	
TIM_TRGO_ENABLE	
TIM_TRGO_UPDATE	
TIM_TRGO_OC1	
TIM_TRGO_OC1REF	
TIM_TRGO_OC2REF	
TIM_TRGO_OC3REF	
TIM_TRGO_OC4REF	
<b><i>Master slave mode</i></b>	
TIM_MASTERSLAVEMODE_ENABLE	
TIM_MASTERSLAVEMODE_DISABLE	
<b><i>One pulse mode</i></b>	
TIM_OPMODE_SINGLE	
TIM_OPMODE_REPETITIVE	
<b><i>Output compare and PWM modes</i></b>	
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

***Output compare N state***

TIM\_OUTPUTNSTATE\_DISABLE  
TIM\_OUTPUTNSTATE\_ENABLE

***Output compare polarity***

TIM\_OCPOLARITY\_HIGH  
TIM\_OCPOLARITY\_LOW

***Output compare state***

TIM\_OUTPUTSTATE\_DISABLE  
TIM\_OUTPUTSTATE\_ENABLE

***Output fast state***

TIM\_OCFAST\_DISABLE  
TIM\_OCFAST\_ENABLE

***Slave mode***

TIM\_SLAVEMODE\_DISABLE  
TIM\_SLAVEMODE\_RESET  
TIM\_SLAVEMODE\_GATED  
TIM\_SLAVEMODE\_TRIGGER  
TIM\_SLAVEMODE\_EXTERNAL1

***TI1 selection***

TIM\_TI1SELECTION\_CH1  
TIM\_TI1SELECTION\_XORCOMBINATION

***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

***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.

***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`

## 46 HAL TIM Extension Driver

### 46.1 TIMEEx Firmware driver registers structures

#### 46.1.1 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\*](#)

### 46.2 TIMEEx Firmware driver API description

#### 46.2.1 TIM specific features integration

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
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

#### 46.2.2 How to use this driver

1. Enable the TIM interface clock using `__HAL_RCC_TIMx_CLK_ENABLE()`;
2. 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 operating mode using one of the configuration function of this driver:

- `HAL_TIMEx_MasterConfigSynchronization()` to configure the peripheral in master mode.
5. Remap the Timer I/O using `HAL_TIMEx_RemapConfig()` API.

### 46.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure Master and the Slave synchronization.

This section contains the following APIs:

- `HAL\_TIMEx\_MasterConfigSynchronization\(\)`
- `HAL\_TIMEx\_RemapConfig\(\)`

### 46.2.4 Detailed description of functions

#### `HAL_TIMEx_RemapConfig`

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)</code>
Function Description	Configures the remapping of the TIM2, TIM3, TIM21 and TIM22 inputs.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>Remap:</b> specifies the TIM input remapping source. This parameter is a combination of the following values depending on TIM instance:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• It is not possible to connect TIM2 and TIM21 on <code>GPIOB5_AF4</code> at the same time. When selecting <code>TIM3_TI2_GPIOB5_AF4</code>, Channel2 of TIM3 will be connected to <code>GPIOB5_AF4</code> and Channel2 of TIM22 will be connected to some other GPIOs. (refer to alternate functions for more details) When selecting <code>TIM3_TI2_GPIO_DEF</code>, Channel2 of Timer 3 will be connected an GPIO (other than <code>GPIOB5_AF4</code>) and Channel2 of TIM22 will be connected to <code>GPIOB5_AF4</code>.</li> </ul>

#### `HAL_TIMEx_MasterConfigSynchronization`

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)</code>
Function Description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM handle.</li> <li>• <b>sMasterConfig:</b> pointer to a <code>TIM_MasterConfigTypeDef</code> structure that contains the selected trigger output (TRGO) and the Master/Slave mode.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

## 46.3 TIME<sub>x</sub> Firmware driver defines

### 46.3.1 TIME<sub>x</sub>

#### *Remapping*

TIM2\_ETR\_GPIO  
TIM2\_ETR\_HSI48  
TIM2\_ETR\_HSI16  
TIM2\_ETR\_LSE  
TIM2\_ETR\_COMP2\_OUT  
TIM2\_ETR\_COMP1\_OUT  
TIM2\_TI4\_GPIO  
TIM2\_TI4\_COMP2  
TIM2\_TI4\_COMP1  
TIM21\_ETR\_GPIO  
TIM21\_ETR\_COMP2\_OUT  
TIM21\_ETR\_COMP1\_OUT  
TIM21\_ETR\_LSE  
TIM21\_TI1\_GPIO  
TIM21\_TI1\_MCO  
TIM21\_TI1\_RTC\_WKUT\_IT  
TIM21\_TI1\_HSE\_RTC  
TIM21\_TI1\_MSI  
TIM21\_TI1\_LSE  
TIM21\_TI1\_LSI  
TIM21\_TI1\_COMP1\_OUT  
TIM21\_TI2\_GPIO  
TIM21\_TI2\_COMP2\_OUT  
TIM22\_ETR\_LSE  
TIM22\_ETR\_COMP2\_OUT  
TIM22\_ETR\_COMP1\_OUT  
TIM22\_ETR\_GPIO  
TIM22\_TI1\_GPIO1  
TIM22\_TI1\_COMP2\_OUT  
TIM22\_TI1\_COMP1\_OUT  
TIM22\_TI1\_GPIO2  
TIM3\_TI4\_GPIO\_DEF  
TIM3\_TI4\_GPIOC9\_AF2

TIM3\_TI2\_GPIO\_DEF  
TIM3\_TI2\_GPIOB5\_AF4  
TIM3\_TI1\_USB\_SOF  
TIM3\_TI1\_GPIO  
TIM3\_ETR\_GPIO  
TIM3\_ETR\_HSI  
IS\_TIM\_REMAP  
IS\_CHANNEL\_AVAILABLE  
***Trigger 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  
IS\_TIM\_TRGO\_SOURCE

## 47 HAL TSC Generic Driver

### 47.1 TSC Firmware driver registers structures

#### 47.1.1 TSC\_InitTypeDef

##### Data Fields

- *uint32\_t CTPulseHighLength*
- *uint32\_t CTPulseLowLength*
- *uint32\_t SpreadSpectrum*
- *uint32\_t SpreadSpectrumDeviation*
- *uint32\_t SpreadSpectrumPrescaler*
- *uint32\_t PulseGeneratorPrescaler*
- *uint32\_t MaxCountValue*
- *uint32\_t IODefaultMode*
- *uint32\_t SynchroPinPolarity*
- *uint32\_t AcquisitionMode*
- *uint32\_t MaxCountInterrupt*
- *uint32\_t ChannelIOs*
- *uint32\_t ShieldIOs*
- *uint32\_t SamplingIOs*

##### Field Documentation

- ***uint32\_t TSC\_InitTypeDef::CTPulseHighLength***  
Charge-transfer high pulse length
- ***uint32\_t TSC\_InitTypeDef::CTPulseLowLength***  
Charge-transfer low pulse length
- ***uint32\_t TSC\_InitTypeDef::SpreadSpectrum***  
Spread spectrum activation
- ***uint32\_t TSC\_InitTypeDef::SpreadSpectrumDeviation***  
Spread spectrum deviation
- ***uint32\_t TSC\_InitTypeDef::SpreadSpectrumPrescaler***  
Spread spectrum prescaler
- ***uint32\_t TSC\_InitTypeDef::PulseGeneratorPrescaler***  
Pulse generator prescaler
- ***uint32\_t TSC\_InitTypeDef::MaxCountValue***  
Max count value
- ***uint32\_t TSC\_InitTypeDef::IODefaultMode***  
IO default mode
- ***uint32\_t TSC\_InitTypeDef::SynchroPinPolarity***  
Synchro pin polarity
- ***uint32\_t TSC\_InitTypeDef::AcquisitionMode***  
Acquisition mode
- ***uint32\_t TSC\_InitTypeDef::MaxCountInterrupt***  
Max count interrupt activation
- ***uint32\_t TSC\_InitTypeDef::ChannelIOs***  
Channel IOs mask

- *uint32\_t TSC\_InitTypeDef::ShieldIOs*  
Shield IOs mask
- *uint32\_t TSC\_InitTypeDef::SamplingIOs*  
Sampling IOs mask

### 47.1.2 TSC\_IOConfigTypeDef

#### Data Fields

- *uint32\_t ChannelIOs*
- *uint32\_t ShieldIOs*
- *uint32\_t SamplingIOs*

#### Field Documentation

- *uint32\_t TSC\_IOConfigTypeDef::ChannelIOs*  
Channel IOs mask
- *uint32\_t TSC\_IOConfigTypeDef::ShieldIOs*  
Shield IOs mask
- *uint32\_t TSC\_IOConfigTypeDef::SamplingIOs*  
Sampling IOs mask

### 47.1.3 TSC\_HandleTypeDef

#### Data Fields

- *TSC\_TypeDef \* Instance*
- *TSC\_InitTypeDef Init*
- *\_IO HAL\_TSC\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

#### Field Documentation

- *TSC\_TypeDef\* TSC\_HandleTypeDef::Instance*  
Register base address
- *TSC\_InitTypeDef TSC\_HandleTypeDef::Init*  
Initialization parameters
- *\_IO HAL\_TSC\_StateTypeDef TSC\_HandleTypeDef::State*  
Peripheral state
- *HAL\_LockTypeDef TSC\_HandleTypeDef::Lock*  
Lock feature

## 47.2 TSC Firmware driver API description

### 47.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

#### 47.2.2 How to use this driver

1. Enable the TSC interface clock using `__HAL_RCC_TSC_CLK_ENABLE()` macro.
2. GPIO pins configuration
  - Enable the clock for the TSC GPIOs using `__HAL_RCC_GPIOx_CLK_ENABLE()` macro.
  - Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using `HAL_GPIO_Init()` function.
  - Configure the alternate function on all the TSC pins using `HAL_xxxx()` function.
3. Interrupts configuration
  - Configure the NVIC (if the interrupt model is used) using `HAL_xxx()` function.
4. TSC configuration
  - Configure all TSC parameters and used TSC IOs using `HAL_TSC_Init()` function.

#### Acquisition sequence

- Discharge all IOs using `HAL_TSC_IODischarge()` function.
- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using `HAL_TSC_IOConfig()` function.
- Launch the acquisition using either `HAL_TSC_Start()` or `HAL_TSC_Start_IT()` function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either `HAL_TSC_PollForAcquisition()` or `HAL_TSC_GetState()` function or using WFI instruction for example.
- Check the group acquisition status using `HAL_TSC_GroupGetStatus()` function.
- Read the acquisition value using `HAL_TSC_GroupGetValue()` function.

#### 47.2.3 IO Operation functions

This section provides functions allowing to:

- Start acquisition in polling mode.
- Start acquisition in interrupt mode.
- Stop conversion in polling mode.
- Stop conversion in interrupt mode.
- Get group acquisition status.

- Get group acquisition value.

This section contains the following APIs:

- `HAL_TSC_Start()`
- `HAL_TSC_Start_IT()`
- `HAL_TSC_Stop()`
- `HAL_TSC_Stop_IT()`
- `HAL_TSC_GroupGetStatus()`
- `HAL_TSC_GroupGetValue()`
- `HAL_TSC_PollForAcquisition()`

#### 47.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs

This section contains the following APIs:

- `HAL_TSC_IOConfig()`
- `HAL_TSC_IODischarge()`

#### 47.2.5 State functions

This subsection provides functions allowing to

- Get TSC state.
- Poll for acquisition completed.
- Handles TSC interrupt request.

This section contains the following APIs:

- `HAL_TSC_GetState()`
- `HAL_TSC_PollForAcquisition()`
- `HAL_TSC_IRQHandler()`
- `HAL_TSC_ConvCpltCallback()`
- `HAL_TSC_ErrorCallback()`

#### 47.2.6 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- `HAL_TSC_Init()`
- `HAL_TSC_DelInit()`
- `HAL_TSC_MspInit()`
- `HAL_TSC_MspDelInit()`

#### 47.2.7 Detailed description of functions

##### `HAL_TSC_Init`

Function Name	<code>HAL_StatusTypeDef HAL_TSC_Init (TSC_HandleTypeDef *htsc)</code>
---------------	---

Function Description	Initializes the TSC peripheral according to the specified parameters in the TSC_InitTypeDef structure.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> TSC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TSC_DeInit</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef * htsc)</b>
Function Description	Deinitializes the TSC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> TSC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TSC_MspInit</b>	
Function Name	<b>void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)</b>
Function Description	Initializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
<b>HAL_TSC_MspDelInit</b>	
Function Name	<b>void HAL_TSC_MspDelInit (TSC_HandleTypeDef * htsc)</b>
Function Description	Deinitializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
<b>HAL_TSC_Start</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TSC_Start (TSC_HandleTypeDef * htsc)</b>
Function Description	Starts the acquisition.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
<b>HAL_TSC_Start_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)</b>
Function Description	Enables the interrupt and starts the acquisition.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified TSC.

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

### HAL\_TSC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)</b>
Function Description	Stops the acquisition previously launched in polling mode.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_TSC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)</b>
Function Description	Stops the acquisition previously launched in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_TSC\_PollForAcquisition

Function Name	<b>HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)</b>
Function Description	Start acquisition and wait until completion.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> state</li> </ul>

Notes

- There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

### HAL\_TSC\_GroupGetStatus

Function Name	<b>TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (TSC_HandleTypeDef * htsc, uint32_t gx_index)</b>
Function Description	Gets the acquisition status for a group.
Parameters	<ul style="list-style-type: none"> <li><b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li><b>gx_index:</b> Index of the group</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Group:</b> status</li> </ul>

### HAL\_TSC\_GroupGetValue

Function Name	<b>uint32_t HAL_TSC_GroupGetValue (TSC_HandleTypeDef * htsc, uint32_t gx_index)</b>
---------------	---

---

Function Description	Gets the acquisition measure for a group.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>gx_index:</b> Index of the group</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Acquisition:</b> measure</li> </ul>

### HAL\_TSC\_IOConfig

Function Name	<b>HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)</b>
Function Description	Configures TSC IOs.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>config:</b> pointer to the configuration structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TSC\_IODischarge

Function Name	<b>HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)</b>
Function Description	Discharge TSC IOs.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>choice:</b> enable or disable</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_TSC\_GetState

Function Name	<b>HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)</b>
Function Description	Return the TSC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

### HAL\_TSC\_IRQHandler

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

### HAL\_TSC\_ConvCpltCallback

Function Name	<b>void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef *</b>
---------------	---

**htsc)**

Function Description	Acquisition completed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

**HAL\_TSC\_ErrorCallback**

Function Name	<b>void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)</b>
Function Description	Error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htsc:</b> pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

## 47.3 TSC Firmware driver defines

### 47.3.1 TSC

***TSC Exported Constants***

TSC\_CTPH\_1CYCLE  
TSC\_CTPH\_2CYCLES  
TSC\_CTPH\_3CYCLES  
TSC\_CTPH\_4CYCLES  
TSC\_CTPH\_5CYCLES  
TSC\_CTPH\_6CYCLES  
TSC\_CTPH\_7CYCLES  
TSC\_CTPH\_8CYCLES  
TSC\_CTPH\_9CYCLES  
TSC\_CTPH\_10CYCLES  
TSC\_CTPH\_11CYCLES  
TSC\_CTPH\_12CYCLES  
TSC\_CTPH\_13CYCLES  
TSC\_CTPH\_14CYCLES  
TSC\_CTPH\_15CYCLES  
TSC\_CTPH\_16CYCLES  
TSC\_CTPL\_1CYCLE  
TSC\_CTPL\_2CYCLES  
TSC\_CTPL\_3CYCLES  
TSC\_CTPL\_4CYCLES  
TSC\_CTPL\_5CYCLES

TSC\_CTPL\_6CYCLES  
TSC\_CTPL\_7CYCLES  
TSC\_CTPL\_8CYCLES  
TSC\_CTPL\_9CYCLES  
TSC\_CTPL\_10CYCLES  
TSC\_CTPL\_11CYCLES  
TSC\_CTPL\_12CYCLES  
TSC\_CTPL\_13CYCLES  
TSC\_CTPL\_14CYCLES  
TSC\_CTPL\_15CYCLES  
TSC\_CTPL\_16CYCLES  
TSC\_SS\_PRESC\_DIV1  
TSC\_SS\_PRESC\_DIV2  
TSC\_PG\_PRESC\_DIV1  
TSC\_PG\_PRESC\_DIV2  
TSC\_PG\_PRESC\_DIV4  
TSC\_PG\_PRESC\_DIV8  
TSC\_PG\_PRESC\_DIV16  
TSC\_PG\_PRESC\_DIV32  
TSC\_PG\_PRESC\_DIV64  
TSC\_PG\_PRESC\_DIV128  
TSC\_MCV\_255  
TSC\_MCV\_511  
TSC\_MCV\_1023  
TSC\_MCV\_2047  
TSC\_MCV\_4095  
TSC\_MCV\_8191  
TSC\_MCV\_16383  
TSC\_IODEF\_OUT\_PP\_LOW  
TSC\_IODEF\_IN\_FLOAT  
TSC\_SYNC\_POLARITY\_FALLING  
TSC\_SYNC\_POLARITY\_RISING  
TSC\_ACQ\_MODE\_NORMAL  
TSC\_ACQ\_MODE\_SYNCHRO  
TSC\_IOMODE\_UNUSED  
TSC\_IOMODE\_CHANNEL

TSC\_IOMODE\_SHIELD  
TSC\_IOMODE\_SAMPLING  
TSC\_NB\_OF\_GROUPS  
TSC\_GROUP1  
TSC\_GROUP2  
TSC\_GROUP3  
TSC\_GROUP4  
TSC\_GROUP5  
TSC\_GROUP6  
TSC\_GROUP7  
TSC\_GROUP8  
TSC\_ALL\_GROUPS  
TSC\_GROUP1\_IDX  
TSC\_GROUP2\_IDX  
TSC\_GROUP3\_IDX  
TSC\_GROUP4\_IDX  
TSC\_GROUP5\_IDX  
TSC\_GROUP6\_IDX  
TSC\_GROUP7\_IDX  
TSC\_GROUP8\_IDX  
TSC\_GROUP1\_IO1  
TSC\_GROUP1\_IO2  
TSC\_GROUP1\_IO3  
TSC\_GROUP1\_IO4  
TSC\_GROUP1\_ALL\_IOS  
TSC\_GROUP2\_IO1  
TSC\_GROUP2\_IO2  
TSC\_GROUP2\_IO3  
TSC\_GROUP2\_IO4  
TSC\_GROUP2\_ALL\_IOS  
TSC\_GROUP3\_IO1  
TSC\_GROUP3\_IO2  
TSC\_GROUP3\_IO3  
TSC\_GROUP3\_IO4  
TSC\_GROUP3\_ALL\_IOS  
TSC\_GROUP4\_IO1

TSC\_GROUP4\_IO2  
TSC\_GROUP4\_IO3  
TSC\_GROUP4\_IO4  
TSC\_GROUP4\_ALL\_IOS  
TSC\_GROUP5\_IO1  
TSC\_GROUP5\_IO2  
TSC\_GROUP5\_IO3  
TSC\_GROUP5\_IO4  
TSC\_GROUP5\_ALL\_IOS  
TSC\_GROUP6\_IO1  
TSC\_GROUP6\_IO2  
TSC\_GROUP6\_IO3  
TSC\_GROUP6\_IO4  
TSC\_GROUP6\_ALL\_IOS  
TSC\_GROUP7\_IO1  
TSC\_GROUP7\_IO2  
TSC\_GROUP7\_IO3  
TSC\_GROUP7\_IO4  
TSC\_GROUP7\_ALL\_IOS  
TSC\_GROUP8\_IO1  
TSC\_GROUP8\_IO2  
TSC\_GROUP8\_IO3  
TSC\_GROUP8\_IO4  
TSC\_GROUP8\_ALL\_IOS  
TSC\_ALL\_GROUPS\_ALL\_IOS

**TSC Exported Macros**

`_HAL_TSC_RESET_HANDLE_STATE`

**Description:**

- Reset TSC handle state.

**Parameters:**

- `_HANDLE_`: TSC handle

**Return value:**

- None

`_HAL_TSC_ENABLE`

**Description:**

- Enable the TSC peripheral.

**Parameters:**

- `_HANDLE_`: TSC handle

`__HAL_TSC_DISABLE`

**Return value:**

- None

**Description:**

- Disable the TSC peripheral.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_START_ACQ`

**Description:**

- Start acquisition.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_STOP_ACQ`

**Description:**

- Stop acquisition.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_SET_IODEF_OUTPPLOW`

**Description:**

- Set IO default mode to output push-pull low.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_SET_IODEF_INFLOAT`

**Description:**

- Set IO default mode to input floating.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_SET_SYNC_POL_FALL`

**Description:**

- Set synchronization polarity to falling edge.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_SET_SYNC_POL_RISE_HIGH`

**Description:**

- Set synchronization polarity to rising edge and high level.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None

`__HAL_TSC_ENABLE_IT`

**Description:**

- Enable TSC interrupt.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__INTERRUPT__`: TSC interrupt

**Return value:**

- None

`__HAL_TSC_DISABLE_IT`

**Description:**

- Disable TSC interrupt.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__INTERRUPT__`: TSC interrupt

**Return value:**

- None

`__HAL_TSC_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: TSC Handle
- `__INTERRUPT__`: TSC interrupt

**Return value:**

- SET: or RESET

`__HAL_TSC_GET_FLAG`

**Description:**

- Get the selected TSC's flag status.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__FLAG__`: TSC flag

`_HAL_TSC_CLEAR_FLAG`

**Return value:**

- SET or RESET

**Description:**

- Clear the TSC's pending flag.

**Parameters:**

- `_HANDLE_`: TSC handle
- `_FLAG_`: TSC flag

**Return value:**

- None

`_HAL_TSC_ENABLE_HYSTERESIS`

**Description:**

- Enable schmitt trigger hysteresis on a group of IOs.

**Parameters:**

- `_HANDLE_`: TSC handle
- `_GX_IOY_MASK_`: IOs mask

**Return value:**

- None

`_HAL_TSC_DISABLE_HYSTERESIS`

**Description:**

- Disable schmitt trigger hysteresis on a group of IOs.

**Parameters:**

- `_HANDLE_`: TSC handle
- `_GX_IOY_MASK_`: IOs mask

**Return value:**

- None

`_HAL_TSC_OPEN_ANALOG_SWITCH`

**Description:**

- Open analog switch on a group of IOs.

**Parameters:**

- `_HANDLE_`: TSC handle
- `_GX_IOY_MASK_`: IOs mask

**Return value:**

- None

`_HAL_TSC_CLOSE_ANALOG_SWITCH`

**Description:**

- Close analog switch on a group of IOs.

**Parameters:**

- `_HANDLE_`: TSC handle
- `_GX_IOY_MASK_`: IOs mask

**Return value:**

- None

`__HAL_TSC_ENABLE_CHANNEL`

**Description:**

- Enable a group of IOs in channel mode.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Return value:**

- None

`__HAL_TSC_DISABLE_CHANNEL`

**Description:**

- Disable a group of channel IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Return value:**

- None

`__HAL_TSC_ENABLE_SAMPLING`

**Description:**

- Enable a group of IOs in sampling mode.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Return value:**

- None

`__HAL_TSC_DISABLE_SAMPLING`

**Description:**

- Disable a group of sampling IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Return value:**

- None

`__HAL_TSC_ENABLE_GROUP`

**Description:**

- Enable acquisition groups.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_MASK__`: Groups mask

**Return value:**

- None

`__HAL_TSC_DISABLE_GROUP`**Description:**

- Disable acquisition groups.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_MASK__`: Groups mask

**Return value:**

- None

`__HAL_TSC_GET_GROUP_STATUS`**Description:**

- Gets acquisition group status.

**Parameters:**

- `__HANDLE__`: TSC Handle
- `__GX_INDEX__`: Group index

**Return value:**

- SET: or RESET

***TSC Flags Definition***`TSC_FLAG_EOA``TSC_FLAG_MCE`***TSC Interrupts Definition***`TSC_IT_EOA``TSC_IT_MCE`

## 48 HAL UART Generic Driver

### 48.1 UART Firmware driver registers structures

#### 48.1.1 **UART\_InitTypeDef**

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t HwFlowCtl*
- *uint32\_t OverSampling*
- *uint32\_t OneBitSampling*

##### Field Documentation

- ***uint32\_t UART\_InitTypeDef::BaudRate***  
This member configures the UART communication baud rate. The baud rate register is computed using the following formula:  
If oversampling is 16 or in LIN mode, Baud Rate Register =  $((PCLKx) / ((huart->Init.BaudRate)))$   
If oversampling is 8, Baud Rate Register[15:4] =  $((2 * PCLKx) / ((huart->Init.BaudRate)))$ [15:4]  
Baud Rate Register[3] = 0  
Baud Rate Register[2:0] =  $((2 * PCLKx) / ((huart->Init.BaudRate)))$ [3:0] >> 1
- ***uint32\_t UART\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [\*\*UARTEx\\_Word\\_Length\*\*](#)
- ***uint32\_t UART\_InitTypeDef::StopBits***  
Specifies the number of stop bits transmitted. This parameter can be a value of [\*\*UART\\_Stop\\_Bits\*\*](#)
- ***uint32\_t UART\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of [\*\*UART\\_Parity\*\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t UART\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*\*UART\\_Mode\*\*](#)
- ***uint32\_t UART\_InitTypeDef::HwFlowCtl***  
Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [\*\*UART\\_Hardware\\_Flow\\_Control\*\*](#)
- ***uint32\_t UART\_InitTypeDef::OverSampling***  
Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [\*\*UART\\_Over\\_Sampling\*\*](#)
- ***uint32\_t UART\_InitTypeDef::OneBitSampling***  
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [\*\*UART\\_One\\_Bit\*\*](#)

## 48.1.2 **UART\_AdvFeatureInitTypeDef**

### Data Fields

- *uint32\_t AdvFeatureInit*
- *uint32\_t TxPinLevelInvert*
- *uint32\_t RxPinLevelInvert*
- *uint32\_t DataInvert*
- *uint32\_t Swap*
- *uint32\_t OverrunDisable*
- *uint32\_t DMADisableonRxError*
- *uint32\_t AutoBaudRateEnable*
- *uint32\_t AutoBaudRateMode*
- *uint32\_t MSBFirst*

### Field Documentation

- ***uint32\_t UART\_AdvFeatureInitTypeDef::AdvFeatureInit***  
Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of [\*\*UART\\_Advanced\\_Features\\_Initialization\\_Type\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::TxPinLevelInvert***  
Specifies whether the TX pin active level is inverted. This parameter can be a value of [\*\*UART\\_Tx\\_Inv\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::RxPinLevelInvert***  
Specifies whether the RX pin active level is inverted. This parameter can be a value of [\*\*UART\\_Rx\\_Inv\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::DataInvert***  
Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of [\*\*UART\\_Data\\_Inv\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::Swap***  
Specifies whether TX and RX pins are swapped. This parameter can be a value of [\*\*UART\\_Rx\\_Tx\\_Swap\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::OverrunDisable***  
Specifies whether the reception overrun detection is disabled. This parameter can be a value of [\*\*UART\\_Overrun\\_Disable\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::DMADisableonRxError***  
Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of [\*\*UART\\_DMA\\_Disable\\_on\\_Rx\\_Error\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::AutoBaudRateEnable***  
Specifies whether auto Baud rate detection is enabled. This parameter can be a value of [\*\*UART\\_AutoBaudRate\\_Enable\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::AutoBaudRateMode***  
If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of [\*\*UARTEX\\_AutoBaud\\_Rate\\_Mode\*\*](#)
- ***uint32\_t UART\_AdvFeatureInitTypeDef::MSBFirst***  
Specifies whether MSB is sent first on UART line. This parameter can be a value of [\*\*UART\\_MSB\\_First\*\*](#)

## 48.1.3 **UART\_HandleTypeDef**

### Data Fields

- ***USART\_TypeDef \* Instance***
- ***UART\_InitTypeDef Init***
- ***UART\_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***uint16\_t Mask***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_UART\_StateTypeDef 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
- ***UART\_AdvFeatureInitTypeDef UART\_HandleTypeDef::AdvancedInit***  
UART Advanced Features initialization parameters
- ***uint8\_t\* UART\_HandleTypeDef::pTxBuffPtr***  
Pointer to UART Tx transfer Buffer
- ***uint16\_t UART\_HandleTypeDef::TxXferSize***  
UART Tx Transfer size
- ***uint16\_t UART\_HandleTypeDef::TxXferCount***  
UART Tx Transfer Counter
- ***uint8\_t\* UART\_HandleTypeDef::pRxBuffPtr***  
Pointer to UART Rx transfer Buffer
- ***uint16\_t UART\_HandleTypeDef::RxXferSize***  
UART Rx Transfer size
- ***uint16\_t UART\_HandleTypeDef::RxXferCount***  
UART Rx Transfer Counter
- ***uint16\_t UART\_HandleTypeDef::Mask***  
UART Rx RDR register mask
- ***DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmatx***  
UART Tx DMA Handle parameters
- ***DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmarx***  
UART Rx DMA Handle parameters
- ***HAL\_LockTypeDef UART\_HandleTypeDef::Lock***  
Locking object
- ***\_\_IO HAL\_UART\_StateTypeDef UART\_HandleTypeDef::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

## 48.2 UART Firmware driver API description

### 48.2.1 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.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL\_UART\_Init(), HAL\_HalfDuplex\_Init(), HAL\_LIN\_Init() and HAL\_MultiProcessorEx\_Init() API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

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\(\)`](#)

### 48.2.2 IO operation functions

This section contains the following APIs:

- [`HAL\_UART\_Transmit\(\)`](#)
- [`HAL\_UART\_Receive\(\)`](#)
- [`HAL\_UART\_Transmit\_IT\(\)`](#)
- [`HAL\_UART\_Receive\_IT\(\)`](#)
- [`HAL\_UART\_Transmit\_DMA\(\)`](#)
- [`HAL\_UART\_Receive\_DMA\(\)`](#)

- [`HAL\_UART\_DMAPause\(\)`](#)
- [`HAL\_UART\_DMAResume\(\)`](#)
- [`HAL\_UART\_DMAStop\(\)`](#)
- [`HAL\_UART\_IRQHandler\(\)`](#)
- [`HAL\_UART\_TxCpltCallback\(\)`](#)
- [`HAL\_UART\_TxHalfCpltCallback\(\)`](#)
- [`HAL\_UART\_RxCpltCallback\(\)`](#)
- [`HAL\_UART\_RxHalfCpltCallback\(\)`](#)
- [`HAL\_UART\_ErrorCallback\(\)`](#)

### 48.2.3 Peripheral Control functions

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

- `HAL_MultiProcessor_EnableMuteMode()` API enables mute mode
- `HAL_MultiProcessor_DisableMuteMode()` API disables mute mode
- `HAL_MultiProcessor_EnterMuteMode()` API enters mute mode
- `HAL_HalfDuplex_EnableTransmitter()` API enables the transmitter
- `HAL_HalfDuplex_EnableReceiver()` API enables the receiver
- `HAL_UART_GetState()` API is helpful to check in run-time the state of the UART peripheral
- `HAL_UART_GetError()` API is helpful to check in run-time the error state of the UART peripheral

This section contains the following APIs:

- [`HAL\_MultiProcessor\_EnableMuteMode\(\)`](#)
- [`HAL\_MultiProcessor\_DisableMuteMode\(\)`](#)
- [`HAL\_MultiProcessor\_EnterMuteMode\(\)`](#)
- [`HAL\_HalfDuplex\_EnableTransmitter\(\)`](#)
- [`HAL\_HalfDuplex\_EnableReceiver\(\)`](#)
- [`HAL\_LIN\_SendBreak\(\)`](#)
- [`HAL\_UART\_GetState\(\)`](#)
- [`HAL\_UART\_GetError\(\)`](#)

### 48.2.4 Detailed description of functions

#### `HAL_UART_Init`

Function Name	<code>HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)</code>
Function Description	Initializes the UART mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### `HAL_HalfDuplex_Init`

Function Name	<code>HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)</code>
Function Description	Initializes the half-duplex mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .

Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

**HAL\_LIN\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)</b>
Function Description	Initializes the LIN mode according to the specified parameters in the <b>UART_InitTypeDef</b> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> <li><b>BreakDetectLength:</b> specifies the LIN break detection length. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>UART_LINBREAKDETECTLENGTH_10B:</b> 10-bit break detection</li> <li>– <b>UART_LINBREAKDETECTLENGTH_11B:</b> 11-bit break detection</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

**HAL\_MultiProcessor\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)</b>
Function Description	Initializes the multiprocessor mode according to the specified parameters in the <b>UART_InitTypeDef</b> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> UART handle</li> <li><b>Address:</b> UART node address (4-, 6-, 7- or 8-bit long)</li> <li><b>WakeUpMethod:</b> specifies the UART wakeup method. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>UART_WAKEUPMETHOD_IDLELINE:</b> WakeUp by an idle line detection</li> <li>– <b>UART_WAKEUPMETHOD_ADDRESSMARK:</b> WakeUp by an address mark</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.</li> <li>If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API <b>HAL_MultiProcessorEx_AddressLength_Set()</b> must be called after <b>HAL_MultiProcessor_Init()</b>.</li> </ul>

**HAL\_UART\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)</b>
Function Description	Deinitializes the UART peripheral.

---

Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_UART\_MspInit

Function Name	<b>void HAL_UART_MspInit (UART_HandleTypeDef * huart)</b>
Function Description	UART MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_UART\_MspDeInit

Function Name	<b>void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)</b>
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

### HAL\_UART\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</li><li>• <b>pData:</b> pointer to data buffer</li><li>• <b>Size:</b> amount of data to be sent</li><li>• <b>Timeout:</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_UART\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</li><li>• <b>pData:</b> pointer to data buffer</li><li>• <b>Size:</b> amount of data to be received</li><li>• <b>Timeout:</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL:</b> status</li></ul>

### HAL\_UART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in interrupt mode.

Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_UART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_UART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_UART\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the UART parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

### HAL\_UART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

**HAL\_UART\_DMAResume**

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_UART\_DMAStop**

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_UART\_IRQHandler**

Function Name	<b>void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)</b>
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_UART\_TxHalfCpltCallback**

Function Name	<b>void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_UART\_TxCpltCallback**

Function Name	<b>void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_UART\_RxHalfCpltCallback**

Function Name	<b>void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>

Return values

- **None:**

### **HAL\_UART\_RxCpltCallback**

Function Name

**void HAL\_UART\_RxCpltCallback (UART\_HandleTypeDef \* huart)**

Function Description

Rx Transfer completed callbacks.

Parameters

- **huart:** uart handle

Return values

- **None:**

### **HAL\_UART\_ErrorCallback**

Function Name

**void HAL\_UART\_ErrorCallback (UART\_HandleTypeDef \* huart)**

Function Description

UART error callbacks.

Parameters

- **huart:** uart handle

Return values

- **None:**

### **HAL\_MultiProcessor\_EnableMuteMode**

Function Name

**HAL\_StatusTypeDef HAL\_MultiProcessor\_EnableMuteMode (UART\_HandleTypeDef \* huart)**

Function Description

Enable UART in mute mode (doesn't mean UART enters mute mode; to enter mute mode, **HAL\_MultiProcessor\_EnterMuteMode()** API must be called)

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

### **HAL\_MultiProcessor\_DisableMuteMode**

Function Name

**HAL\_StatusTypeDef HAL\_MultiProcessor\_DisableMuteMode (UART\_HandleTypeDef \* huart)**

Function Description

Disable UART mute mode (doesn't mean it actually wakes up the software, as it may not have been in mute mode at this very moment).

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

### **HAL\_MultiProcessor\_EnterMuteMode**

Function Name

**void HAL\_MultiProcessor\_EnterMuteMode (UART\_HandleTypeDef \* huart)**

Function Description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

**HAL\_HalfDuplex\_EnableTransmitter**

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)</b>
Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> <li>• <b>None:</b></li> </ul>

**HAL\_HalfDuplex\_EnableReceiver**

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)</b>
Function Description	Enables the UART receiver and disables the UART transmitter.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_LIN\_SendBreak**

Function Name	<b>HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)</b>
Function Description	Transmits break characters.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_UART\_GetState**

Function Name	<b>HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)</b>
Function Description	return the UART state
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> state</li> </ul>

**HAL\_UART\_GetError**

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

**UART\_SetConfig**

Function Name	<b>void UART_SetConfig (UART_HandleTypeDef * huart)</b>
---------------	---

Function Description	Configure the UART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**UART\_CheckIdleState**

Function Name	<b>HAL_StatusTypeDef</b> <b>UART_CheckIdleState</b> <b>(UART_HandleTypeDef * huart)</b>
Function Description	Check the UART Idle State.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**UART\_WaitOnFlagUntilTimeout**

Function Name	<b>HAL_StatusTypeDef</b> <b>UART_WaitOnFlagUntilTimeout</b> <b>(UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Timeout)</b>
Function Description	This function handles UART Communication Timeout.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> <li>• <b>Flag:</b> specifies the UART flag to check.</li> <li>• <b>Status:</b> The new Flag status (SET or RESET).</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**UART\_AdvFeatureConfig**

Function Name	<b>void</b> <b>UART_AdvFeatureConfig</b> <b>(UART_HandleTypeDef * huart)</b>
Function Description	Configure the UART peripheral advanced feautures.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**48.3 UART Firmware driver defines****48.3.1 UART*****UART advanced features initialization type definition***

UART\_ADVFEATURE\_NO\_INIT  
 UART\_ADVFEATURE\_TXINVERT\_INIT  
 UART\_ADVFEATURE\_RXINVERT\_INIT  
 UART\_ADVFEATURE\_DATAINVERT\_INIT  
 UART\_ADVFEATURE\_SWAP\_INIT  
 UART\_ADVFEATURE\_RXOVERRUNDISABLE\_INIT  
 UART\_ADVFEATURE\_DMADISABLEONERROR\_INIT

UART\_ADVFEATURE\_AUTOBAUDRATE\_INIT  
UART\_ADVFEATURE\_MSBFIRST\_INIT  
IS\_UART\_ADVFEATURE\_INIT  
**UART advanced auto baud rate activation definition**  
UART\_ADVFEATURE\_AUTOBAUDRATE\_DISABLE  
UART\_ADVFEATURE\_AUTOBAUDRATE\_ENABLE  
IS\_UART\_ADVFEATURE\_AUTOBAUDRATE  
**UART CR1 DEAT address lsb position definition**  
UART\_CR1\_DEAT\_ADDRESS\_LSB\_POS  
**UART CR1 DEDT address lsb position definition**  
UART\_CR1\_DEDT\_ADDRESS\_LSB\_POS  
**UART CR2 address lsb position definition**  
UART\_CR2\_ADDRESS\_LSB\_POS  
**UART advanced data inv activation definition**  
UART\_ADVFEATURE\_DATAINV\_DISABLE  
UART\_ADVFEATURE\_DATAINV\_ENABLE  
IS\_UART\_ADVFEATURE\_DATAINV  
**UART advanced DMA on Rx error activation definition**  
UART\_ADVFEATURE\_DMA\_ENABLEONRXERROR  
UART\_ADVFEATURE\_DMA\_DISABLEONRXERROR  
IS\_UART\_ADVFEATURE\_DMAONRXERROR  
**UART DMA Rx definition**  
UART\_DMA\_RX\_DISABLE  
UART\_DMA\_RX\_ENABLE  
IS\_UART\_DMA\_RX  
**UART DMA Tx definition**  
UART\_DMA\_TX\_DISABLE  
UART\_DMA\_TX\_ENABLE  
IS\_UART\_DMA\_TX  
**UART driver polarity level definition**  
UART\_DE\_POLARITY\_HIGH  
UART\_DE\_POLARITY\_LOW  
IS\_UART\_DE\_POLARITY  
**UART error definition**  
HAL\_UART\_ERROR\_NONE No error  
HAL\_UART\_ERROR\_PE Parity error

<code>HAL_UART_ERROR_NE</code>	Noise error
<code>HAL_UART_ERROR_FE</code>	frame error
<code>HAL_UART_ERROR_ORE</code>	Overrun error
<code>HAL_UART_ERROR_DMA</code>	DMA transfer error

***UART Exported Macros***

<code>_HAL_UART_RESET_HANDLE_STAT</code>	<b>Description:</b> • Reset UART handle state.
<b>Parameters:</b>	
	• <code>_HANDLE_</code> : specifies the UART Handle. The Handle Instance which can be USART1, USART2 or LPUART.
<code>_HAL_UART_FLUSH_DRREGISTER</code>	<b>Return value:</b> • None
	<b>Description:</b> • Flush the UART Data registers.
	<b>Parameters:</b>
	• <code>_HANDLE_</code> : specifies the UART Handle.
<code>_HAL_UART_CLEAR_FLAG</code>	<b>Description:</b> • Clears the specified UART pending flag.
	<b>Parameters:</b>
	• <code>_HANDLE_</code> : specifies the UART Handle.
	• <code>_FLAG_</code> : specifies the flag to check. This parameter can be any combination of the following values: – <code>UART_CLEAR_PEF</code> – <code>UART_CLEAR_FEF</code> – <code>UART_CLEAR_NEF</code> – <code>UART_CLEAR_OREF</code> – <code>UART_CLEAR_IDLEF</code> – <code>UART_CLEAR_TCF</code> – <code>UART_CLEAR_LBDF</code> – <code>UART_CLEAR_CTSF</code> – <code>UART_CLEAR_RTOF</code> – <code>UART_CLEAR_EOBF</code> – <code>UART_CLEAR_CMF</code> – <code>UART_CLEAR_WUF</code>
	<b>Return value:</b> • None
<code>_HAL_UART_CLEAR_PEFLAG</code>	<b>Description:</b> • Clear the UART PE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- None

`_HAL_UART_CLEAR_FEFLAG`

**Description:**

- Clear the UART FE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- None

`_HAL_UART_CLEAR_NEFLAG`

**Description:**

- Clear the UART NE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- None

`_HAL_UART_CLEAR_OREFLAG`

**Description:**

- Clear the UART ORE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- None

`_HAL_UART_CLEAR_IDLEFLAG`

**Description:**

- Clear the UART IDLE pending flag.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- None

`_HAL_UART_GET_FLAG`

**Description:**

- Checks whether the specified UART flag is set or not.

**Parameters:**

- `_HANDLE_`: specifies the UART

Handle. This parameter can be USART1, USART2 or LPUART.

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `UART_FLAG_RXACK`: Receive enable acknowledge flag
  - `UART_FLAG_TEACK`: Transmit enable acknowledge flag
  - `UART_FLAG_WUF`: Wake up from stop mode flag
  - `UART_FLAG_RWU`: Receiver wake up flag (is the UART in mute mode)
  - `UART_FLAG_SBKF`: Send Break flag
  - `UART_FLAG_CMF`: Character match flag
  - `UART_FLAG_BUSY`: Busy flag
  - `UART_FLAG_ABRF`: Auto Baud rate detection flag
  - `UART_FLAG_ABRE`: Auto Baud rate detection error flag
  - `UART_FLAG_EOBF`: End of block flag
  - `UART_FLAG_RTOF`: Receiver timeout flag
  - `UART_FLAG_CTS`: CTS Change flag (not available for USART4 and USART5)
  - `UART_FLAG_LBD`: LIN Break detection flag
  - `UART_FLAG_TXE`: Transmit data register empty flag
  - `UART_FLAG_TC`: Transmission Complete flag
  - `UART_FLAG_RXNE`: Receive data register not empty flag
  - `UART_FLAG_IDLE`: Idle Line detection flag
  - `UART_FLAG_ORE`: OverRun Error flag
  - `UART_FLAG_NE`: Noise Error flag
  - `UART_FLAG_FE`: Framing Error flag
  - `UART_FLAG_PE`: Parity Error flag

#### Return value:

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

### `__HAL_UART_ENABLE_IT`

#### Description:

- Enables the specified UART interrupt.

#### Parameters:

- `__HANDLE__`: specifies the UART

Handle. This parameter can be USART1, USART2 or LPUART.

- \_\_INTERRUPT\_\_: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - UART\_IT\_PE: Parity Error interrupt
  - UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

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

**Description:**

- Disables the specified UART interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- \_\_INTERRUPT\_\_: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt

- UART\_IT\_IDLE: Idle line detection interrupt
- UART\_IT\_PE: Parity Error interrupt
- UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

`__HAL_UART_GET_IT`

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

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__IT__`: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt (not available for USART4 and USART5)
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - UART\_IT\_ORE: OverRun Error interrupt
  - UART\_IT\_NE: Noise Error interrupt
  - UART\_IT\_FE: Framing Error interrupt
  - UART\_IT\_PE: Parity Error interrupt

**Return value:**

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

`__HAL_UART_GET_IT_SOURCE`

**Description:**

- Checks whether the specified UART interrupt source is enabled.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__IT__`: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - `UART_IT_CTS`: CTS change interrupt (not available for UART4 and UART5)
  - `UART_IT_LBD`: LIN Break detection interrupt
  - `UART_IT_TXE`: Transmit Data Register empty interrupt
  - `UART_IT_TC`: Transmission complete interrupt
  - `UART_IT_RXNE`: Receive Data register not empty interrupt
  - `UART_IT_IDLE`: Idle line detection interrupt
  - `UART_IT_ORE`: OverRun Error interrupt
  - `UART_IT_NE`: Noise Error interrupt
  - `UART_IT_FFE`: Framing Error interrupt
  - `UART_IT_PEF`: Parity Error interrupt

**Return value:**

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

[\\_\\_HAL\\_UART\\_CLEAR\\_IT](#)**Description:**

- Clears the specified UART ISR flag, in setting the proper ICR register flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__IT_CLEAR__`: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - `UART_CLEAR_PEF`: Parity Error Clear Flag
  - `UART_CLEAR_FEF`: Framing Error Clear Flag
  - `UART_CLEAR_NEF`: Noise detected Clear Flag
  - `UART_CLEAR_OREF`: OverRun Error Clear Flag
  - `UART_CLEAR_IDLEF`: IDLE line detected Clear Flag
  - `UART_CLEAR_TCF`: Transmission

- Complete Clear Flag
- UART\_CLEAR\_LBDF: LIN Break Detection Clear Flag
- UART\_CLEAR\_CTSF: CTS Interrupt Clear Flag
- UART\_CLEAR\_RTOF: Receiver Time Out Clear Flag
- UART\_CLEAR\_EOBF: End Of Block Clear Flag
- UART\_CLEAR\_CMF: Character Match Clear Flag
- UART\_CLEAR\_WUF: Wake Up from stop mode Clear Flag

**Return value:**

- None

[\\_\\_HAL\\_UART\\_SEND\\_REQ](#)**Description:**

- Set a specific UART request flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- [\\_\\_REQ\\_\\_](#): specifies the request flag to set. This parameter can be one of the following values:
  - [UART\\_AUTOBAUD\\_REQUEST](#): Auto-Baud Rate Request
  - [UART\\_SENDBREAK\\_REQUEST](#): Send Break Request
  - [UART\\_MUTE\\_MODE\\_REQUEST](#): Mute Mode Request
  - [UART\\_RXDATA\\_FLUSH\\_REQUEST](#): Receive Data flush Request
  - [UART\\_TXDATA\\_FLUSH\\_REQUEST](#): Transmit data flush Request

**Return value:**

- None

[\\_\\_HAL\\_UART\\_ONE\\_BIT\\_SAMPLE\\_ENA  
BLE](#)**Description:**

- Enables the UART one bit sample method.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the UART Handle.

**Return value:**

- None

[\\_\\_HAL\\_UART\\_ONE\\_BIT\\_SAMPLE\\_DIS  
ABLE](#)**Description:**

- Disables the UART one bit sample

method.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_ENABLE`

**Description:**

- Enable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

**Return value:**

- None

`__HAL_UART_DISABLE`

**Description:**

- Disable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

**Return value:**

- None

`__HAL_UART_HWCONTROL_CTS_ENABLE`

**Description:**

- Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

**Return value:**

- None

**Notes:**

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : USART instance should have already been initialised (through call of `HAL_UART_Init()`)macro could only be

called when corresponding UART instance is disabled (i.e  
\_\_HAL\_UART\_DISABLE(\_\_HANDLE\_\_))  
and should be followed by an Enable macro (i.e  
\_\_HAL\_UART\_ENABLE(\_\_HANDLE\_\_)).

### \_\_HAL\_UART\_HWCONTROL\_CTS\_DISABLE

#### Description:

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

#### Parameters:

- \_\_HANDLE\_\_: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

#### Return value:

- None

#### Notes:

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL\_UART\_Init() )macro could only be called when corresponding UART instance is disabled (i.e \_\_HAL\_UART\_DISABLE(\_\_HANDLE\_\_)) and should be followed by an Enable macro (i.e \_\_HAL\_UART\_ENABLE(\_\_HANDLE\_\_)).

### \_\_HAL\_UART\_HWCONTROL\_RTS\_ENABLE

#### Description:

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

#### Parameters:

- \_\_HANDLE\_\_: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

#### Return value:

- None

#### Notes:

- As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART

instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL\_UART\_Init() )macro could only be called when corresponding UART instance is disabled (i.e \_\_HAL\_UART\_DISABLE(\_\_HANDLE\_\_)) and should be followed by an Enable macro (i.e \_\_HAL\_UART\_ENABLE(\_\_HANDLE\_\_)).

#### `__HAL_UART_HWCONTROL_RTS_DISABLE`

##### **Description:**

- Disable RTS flow control This macro allows to disable RTS hardware flow control for a given UART instance, without need to call

##### **Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

##### **Return value:**

- None

##### **Notes:**

- As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : 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_ONE_BIT_ENABLE`

##### **Description:**

- macros to enable the UART's one bit sampling method

##### **Parameters:**

- `__HANDLE__`: specifies the UART Handle.

##### **Return value:**

- None

#### `__HAL_UART_ONE_BIT_DISABLE`

##### **Description:**

- macros to disable the UART's one bit

sampling method

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- None

**Description:**

- BRR division operation to set BRR register with LPUART.

**Parameters:**

- `_PCLK_`: LPUART clock
- `_BAUD_`: Baud rate set by the user

**Return value:**

- Division: result

**Description:**

- BRR division operation to set BRR register in 8-bit oversampling mode.

**Parameters:**

- `_PCLK_`: UART clock
- `_BAUD_`: Baud rate set by the user

**Return value:**

- Division: result

**Description:**

- BRR division operation to set BRR register in 16-bit oversampling mode.

**Parameters:**

- `_PCLK_`: UART clock
- `_BAUD_`: Baud rate set by the user

**Return value:**

- Division: result

**Description:**

- Check whether or not UART instance is Low Power UART.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

**Return value:**

- SET: (instance is LPUART) or RESET (instance isn't LPUART)

IS_UART_BAUDRATE	<b>Description:</b> <ul style="list-style-type: none"><li>Check UART Baud rate.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>BAUDRATE: Baudrate specified by the user. The maximum Baud Rate is derived from the maximum clock on L0 (i.e. 32 MHz) divided by the smallest oversampling used on the USART (i.e. 8)</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>Test: result (TRUE or FALSE).</li></ul>
IS_UART_7B_ADDRESS	<b>Description:</b> <ul style="list-style-type: none"><li>Check UART byte address.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>ADDRESS: UART 8-bit address for wake-up process scheme</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>Test: result (TRUE or FALSE).</li></ul>
IS_UART_4B_ADDRESS	<b>Description:</b> <ul style="list-style-type: none"><li>Check UART 4-bit address.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>ADDRESS: UART 4-bit address for wake-up process scheme</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>Test: result (TRUE or FALSE).</li></ul>
IS_UART_ASSERTIONTIME	<b>Description:</b> <ul style="list-style-type: none"><li>Check UART assertion time.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>TIME: 5-bit value assertion time</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>Test: result (TRUE or FALSE).</li></ul>
IS_UART_DEASSERTIONTIME	<b>Description:</b> <ul style="list-style-type: none"><li>Check UART deassertion time.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>TIME: 5-bit value deassertion time</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>Test: result (TRUE or FALSE).</li></ul>

**UART flags definition**

UART\_FLAG\_RXACK Receive Enable Acknowledge Flag

UART_FLAG_TEACK	Transmit Enable Acknowledge Flag
UART_FLAG_WUF	Wake Up from stop mode Flag
UART_FLAG_RWU	Receive Wake Up from mute mode Flag
UART_FLAG_SBKF	Send Break Flag
UART_FLAG_CMF	Character Match Flag
UART_FLAG_BUSY	Busy Flag
UART_FLAG_ABRF	Auto-Baud Rate Flag
UART_FLAG_ABRE	Auto-Baud Rate Error
UART_FLAG_EOBF	End Of Block Flag
UART_FLAG_RTOF	Receiver Time Out
UART_FLAG_CTS	CTS flag
UART_FLAG_CTSIF	CTS interrupt flag
UART_FLAG_LBDF	LIN Break Detection Flag
UART_FLAG_TXE	Transmit Data Register Empty
UART_FLAG_TC	Transmission Complete
UART_FLAG_RXNE	Read Data Register Not Empty
UART_FLAG_IDLE	IDLE line detected
UART_FLAG_ORE	OverRun Error
UART_FLAG_NE	Noise detected Flag
UART_FLAG_FE	Framing Error
UART_FLAG_PE	Parity Error

***UART half duplex selection definition***

UART\_HALF\_DUPLEX\_DISABLE

UART\_HALF\_DUPLEX\_ENABLE

IS\_UART\_HALF\_DUPLEX

***UART hardware flow control definition***

UART\_HWCONTROL\_NONE

UART\_HWCONTROL\_RTS

UART\_HWCONTROL\_CTS

UART\_HWCONTROL\_RTS\_CTS

IS\_UART\_HARDWARE\_FLOW\_CONTROL

***UART interruption mask definition***

UART\_IT\_MASK

***UART interrupt definition***

UART\_IT\_PE

UART\_IT\_TXE

UART\_IT\_TC  
UART\_IT\_RXNE  
UART\_IT\_IDLE  
UART\_IT\_LBD  
UART\_IT\_CTS  
UART\_IT\_CM  
UART\_IT\_WUF  
UART\_IT\_ERR  
UART\_IT\_ORE  
UART\_IT\_NE  
UART\_IT\_FE

***UART interrupt clear flags definition***

UART\_CLEAR\_PEF Parity Error Clear Flag  
UART\_CLEAR\_FEF Framing Error Clear Flag  
UART\_CLEAR\_NEF Noise detected Clear Flag  
UART\_CLEAR\_OREF OverRun Error Clear Flag  
UART\_CLEAR\_IDLEF IDLE line detected Clear Flag  
UART\_CLEAR\_TCF Transmission Complete Clear Flag  
UART\_CLEAR\_LBDF LIN Break Detection Clear Flag  
UART\_CLEAR\_CTSF CTS Interrupt Clear Flag  
UART\_CLEAR\_RTOF Receiver Time Out Clear Flag  
UART\_CLEAR\_EOBF End Of Block Clear Flag  
UART\_CLEAR\_CMF Character Match Clear Flag  
UART\_CLEAR\_WUF Wake Up from stop mode Clear Flag

***UART LIN enable and disable definition***

UART\_LIN\_DISABLE  
UART\_LIN\_ENABLE  
IS\_UART\_LIN

***UART LIN break detection definition***

UART\_LINBREAKDETECTLENGTH\_10B  
UART\_LINBREAKDETECTLENGTH\_11B  
IS\_UART\_LIN\_BREAK\_DETECT\_LENGTH

***UART mode definition***

UART\_MODE\_RX  
UART\_MODE\_TX  
UART\_MODE\_TX\_RX

IS\_UART\_MODE

**UART advanced MSB first activation definition**

UART\_ADVFEATURE\_MSBFIRST\_DISABLE

UART\_ADVFEATURE\_MSBFIRST\_ENABLE

IS\_UART\_ADVFEATURE\_MSBFIRST

**UART advanced mute mode activation definition**

UART\_ADVFEATURE\_MUTEMODE\_DISABLE

UART\_ADVFEATURE\_MUTEMODE\_ENABLE

IS\_UART\_MUTE\_MODE

**UART one bit definition**

UART\_ONE\_BIT\_SAMPLE\_DISABLE

UART\_ONE\_BIT\_SAMPLE\_ENABLE

IS\_UART\_ONE\_BIT\_SAMPLE

**UART advanced overrun activation definition**

UART\_ADVFEATURE\_OVERRUN\_ENABLE

UART\_ADVFEATURE\_OVERRUN\_DISABLE

IS\_UART\_OVERRUN

**UART over sampling definition**

UART\_OVERSAMPLING\_16

UART\_OVERSAMPLING\_8

IS\_UART\_OVERSAMPLING

**UART parity definition**

UART\_PARITY\_NONE

UART\_PARITY\_EVEN

UART\_PARITY\_ODD

IS\_UART\_PARITY

**UART receiver timeOut definition**

UART\_RECEIVER\_TIMEOUT\_DISABLE

UART\_RECEIVER\_TIMEOUT\_ENABLE

IS\_UART\_RECEIVER\_TIMEOUT

**UART request parameter definition**

UART\_AUTOBAUD\_REQUEST Auto-Baud Rate Request

UART\_SENDBREAK\_REQUEST Send Break Request

UART\_MUTE\_MODE\_REQUEST Mute Mode Request

UART\_RXDATA\_FLUSH\_REQUEST Receive Data flush Request

UART\_TXDATA\_FLUSH\_REQUEST Transmit data flush Request

IS\_UART\_REQUEST\_PARAMETER  
**UART advanced Rx inv activation definition**  
UART\_ADVFEATURE\_RXINV\_DISABLE  
UART\_ADVFEATURE\_RXINV\_ENABLE  
IS\_UART\_ADVFEATURE\_RXINV  
**UART advanced swap activation definition**  
UART\_ADVFEATURE\_SWAP\_DISABLE  
UART\_ADVFEATURE\_SWAP\_ENABLE  
IS\_UART\_ADVFEATURE\_SWAP  
**UART state enable and disable definition**  
UART\_STATE\_DISABLE  
UART\_STATE\_ENABLE  
IS\_UART\_STATE  
**UART stop bit definition**  
UART\_STOPBITS\_1      USART frame with 1 stop bit  
UART\_STOPBITS\_1\_5    USART frame with 1.5 stop bits  
UART\_STOPBITS\_2      USART frame with 2 stop bits  
IS\_UART\_STOPBITS  
IS\_LPUART\_STOPBITS  
**UART advanced stop mode activation definition**  
UART\_ADVFEATURE\_STOPMODE\_DISABLE  
UART\_ADVFEATURE\_STOPMODE\_ENABLE  
IS\_UART\_ADVFEATURE\_STOPMODE  
**UART advanced Tx inv activation definition**  
UART\_ADVFEATURE\_TXINV\_DISABLE  
UART\_ADVFEATURE\_TXINV\_ENABLE  
IS\_UART\_ADVFEATURE\_TXINV  
**UART wake up mode selection definition**  
UART\_WAKEUP\_ON\_ADDRESS  
UART\_WAKEUP\_ON\_STARTBIT  
UART\_WAKEUP\_ON\_READDATA\_NONEMPTY  
IS\_UART\_WAKEUP\_SELECTION

## 49 HAL UART Extension Driver

### 49.1 UARTEEx Firmware driver registers structures

#### 49.1.1 UART\_WakeUpTypeDef

##### Data Fields

- *uint32\_t WakeUpEvent*
- *uint16\_t AddressLength*
- *uint8\_t Address*

##### Field Documentation

- ***uint32\_t UARTEEx\_WakeUpTypeDef::WakeUpEvent***  
Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of [\*UARTEEx\\_WakeUp\\_from\\_Stop\\_Selection\*](#). If set to **UART\_WAKEUP\_ON\_ADDRESS**, the two other fields below must be filled up.
- ***uint16\_t UARTEEx\_WakeUpTypeDef::AddressLength***  
Specifies whether the address is 4 or 7-bit long. This parameter can be a value of [\*UARTEEx\\_WakeUp\\_Address\\_Length\*](#)
- ***uint8\_t UARTEEx\_WakeUpTypeDef::Address***  
UART/USART node address (7-bit long max)

## 49.2 UARTEEx Firmware driver API description

### 49.2.1 Initialization and Configuration functions

The HAL\_RS485Ex\_Init() API follows respectively the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [\*HAL\\_RS485Ex\\_Init\(\)\*](#)

### 49.2.2 Peripheral Control functions

This section provides functions allowing to:

- **UART\_AdvFeatureConfig()** API optionally configures the UART advanced features
- **HAL\_MultiProcessorEx\_AddressLength\_Set()** API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- **HAL\_UARTEEx\_EnableStopMode()** API enables the UART to wake up the MCU from stop mode
- **HAL\_UARTEEx\_DisableStopMode()** API disables the above functionality
- **HAL\_UARTEEx\_EnableClockStopMode()** API enables the UART HSI clock during stop mode
- **HAL\_UARTEEx\_DisableClockStopMode()** API disables the above functionality
- **UART\_Wakeup\_AddressConfig()** API configures the wake-up from stop mode parameters

This section contains the following APIs:

- [\*\*\*HAL\\_UARTEX\\_EnableStopMode\(\)\*\*\*](#)
- [\*\*\*HAL\\_UARTEX\\_EnableClockStopMode\(\)\*\*\*](#)
- [\*\*\*HAL\\_UARTEX\\_DisableStopMode\(\)\*\*\*](#)
- [\*\*\*HAL\\_UARTEX\\_DisableClockStopMode\(\)\*\*\*](#)
- [\*\*\*HAL\\_UARTEX\\_StopModeWakeUpSourceConfig\(\)\*\*\*](#)
- [\*\*\*HAL\\_MultiProcessorEx\\_AddressLength\\_Set\(\)\*\*\*](#)
- [\*\*\*HAL\\_UARTEX\\_WakeupCallback\(\)\*\*\*](#)

### 49.2.3 Detailed description of functions

#### **HAL\_RS485Ex\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)</b>
Function Description	Initializes the RS485 Driver enable feature according to the specified parameters in the <b>UART_InitTypeDef</b> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> <li>• <b>Polarity:</b> select the driver enable polarity This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>UART_DE_POLARITY_HIGH:</b> DE signal is active high</li> <li>– <b>UART_DE_POLARITY_LOW:</b> DE signal is active low</li> </ul> </li> <li>• <b>AssertionTime:</b> Driver Enable assertion time 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)</li> <li>• <b>DeassertionTime:</b> Driver Enable deassertion time 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

#### **HAL\_UARTEX\_StopModeWakeUpSourceConfig**

Function Name	<b>HAL_StatusTypeDef HAL_UARTEX_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)</b>
Function Description	Set Wakeup from Stop mode interrupt flag selection.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle,</li> <li>• <b>WakeUpSelection:</b> address match, Start Bit detection or RXNE bit status. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>UART_WAKEUP_ON_ADDRESS</b></li> <li>– <b>UART_WAKEUP_ON_STARTBIT</b></li> <li>– <b>UART_WAKEUP_ON_READDATA_NONEMPTY</b></li> </ul> </li> </ul>

Return values

- **HAL:** status

### **HAL\_UARTEx\_EnableStopMode**

Function Name

**HAL\_StatusTypeDef HAL\_UARTEx\_EnableStopMode  
(UART\_HandleTypeDef \* huart)**

Function Description

Enable UART Stop Mode The UART is able to wake up the MCU from Stop mode as long as UART clock is HSI or LSE.

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

### **HAL\_UARTEx\_EnableClockStopMode**

Function Name

**HAL\_StatusTypeDef HAL\_UARTEx\_EnableClockStopMode  
(UART\_HandleTypeDef \* huart)**

Function Description

Enable UART Clock in Stop Mode The UART keeps the Clock ON during Stop mode.

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

### **HAL\_UARTEx\_DisableStopMode**

Function Name

**HAL\_StatusTypeDef HAL\_UARTEx\_DisableStopMode  
(UART\_HandleTypeDef \* huart)**

Function Description

Disable UART Stop Mode.

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

### **HAL\_UARTEx\_DisableClockStopMode**

Function Name

**HAL\_StatusTypeDef HAL\_UARTEx\_DisableClockStopMode  
(UART\_HandleTypeDef \* huart)**

Function Description

Disable UART Clock in Stop Mode.

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

### **HAL\_UARTEx\_WakeupCallback**

Function Name

**void HAL\_UARTEx\_WakeupCallback (UART\_HandleTypeDef \* huart)**

Function Description

UART wakeup from Stop mode callback.

Parameters

- **huart:** uart handle

Return values

- **None:**

**HAL\_MultiProcessorEx\_AddressLength\_Set**

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)</b>
Function Description	By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection.

**49.3 UARTE Firmware driver defines****49.3.1 UARTE*****Auto baud rate mode definition***

UART\_ADVFEATURE\_AUTOBAUDRATE\_ONSTARTBIT  
 UART\_ADVFEATURE\_AUTOBAUDRATE\_ONFALLINGEDGE  
 UART\_ADVFEATURE\_AUTOBAUDRATE\_ON0X7FFFRAME  
 UART\_ADVFEATURE\_AUTOBAUDRATE\_ON0X55FRAME  
 IS\_UART\_ADVFEATURE\_AUTOBAUDRATEMODE

***UARTE Exported Macros***

UART_GETCLOCKSOURCE	<b>Description:</b>  <ul style="list-style-type: none"> <li>• Reports the UART clock source.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <u>__HANDLE__</u>: specifies the UART Handle</li> <li>• <u>__CLOCKSOURCE__</u>: output variable</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• UART: clocking source, written in <u>__CLOCKSOURCE__</u>.</li> </ul>
UART_MASK_COMPUTATION	<b>Description:</b>  <ul style="list-style-type: none"> <li>• Reports the UART mask to apply to retrieve the received data according to the word length and to the parity bits activation.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <u>__HANDLE__</u>: specifies the UART Handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• mask: to apply to UART RDR register value.</li> </ul>

***WakeUp address length definition***

UART\_ADDRESS\_DETECT\_4B  
 UART\_ADDRESS\_DETECT\_7B  
 IS\_UART\_ADDRESSLENGTH\_DETECT

***Wakeup methods definition***

UART\_WAKEUPMETHOD\_IDLELINE  
UART\_WAKEUPMETHOD\_ADDRESSMARK  
IS\_UART\_WAKEUPMETHOD  
***Word length definition***  
UART\_WORDLENGTH\_7B  
UART\_WORDLENGTH\_8B  
UART\_WORDLENGTH\_9B  
IS\_UART\_WORD\_LENGTH

## 50 HAL USART Generic Driver

### 50.1 USART Firmware driver registers structures

#### 50.1.1 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: Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
- ***uint32\_t USART\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [\*\*USARTEx\\_Word\\_Length\*\*](#)
- ***uint32\_t USART\_InitTypeDef::StopBits***  
Specifies the number of stop bits transmitted. This parameter can be a value of [\*\*USART\\_Stop\\_Bits\*\*](#)
- ***uint32\_t USART\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of [\*\*USART\\_Parity\*\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t USART\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*\*USART\\_Mode\*\*](#)
- ***uint32\_t USART\_InitTypeDef::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\*\*](#)

## 50.1.2 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**
- **uint16\_t Mask**
- **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
- **uint16\_t USART\_HandleTypeDef::Mask**
- **DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmatx**  
Usart Tx DMA Handle parameters
- **DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmarx**  
Usart Rx DMA Handle parameters
- **HAL\_LockTypeDef USART\_HandleTypeDef::Lock**  
Locking object
- **\_IO HAL\_USART\_StateTypeDef USART\_HandleTypeDef::State**  
Usart communication state
- **\_IO uint32\_t USART\_HandleTypeDef::ErrorCode**  
USART Error code

## 50.2 USART Firmware driver API description

### 50.2.1 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.
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL\_USART\_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual (RM0329)).

This section contains the following APIs:

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

### 50.2.2 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 API's are :
  - HAL\_USART\_Transmit() in simplex mode
  - HAL\_USART\_Receive() in full duplex receive only
  - HAL\_USART\_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
  - HAL\_USART\_Transmit\_IT() in simplex mode
  - HAL\_USART\_Receive\_IT() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_IT() in full duplex mode
  - HAL\_USART\_IRQHandler()
4. No-Blocking mode functions with DMA are :



- HAL\_USART\_Transmit\_DMA() in simplex mode
  - HAL\_USART\_Receive\_DMA() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_DMA() in full duplex mode
  - HAL\_USART\_DMAPause()
  - HAL\_USART\_DMAResume()
  - HAL\_USART\_DMAStop()
5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
- HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_TxHalfCpltCallback()
  - HAL\_USART\_RxHalfCpltCallback()
  - HAL\_USART\_ErrorCallback()
  - HAL\_USART\_TxRxCpltCallback()

This section contains the following APIs:

- [\*\*HAL\\_USART\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TransmitReceive\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TransmitReceive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TransmitReceive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DMAPause\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DMAResume\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DMAStop\(\)\*\*](#)
- [\*\*HAL\\_USART\\_IRQHandler\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TxHalfCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_USART\\_RxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_USART\\_RxHalfCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TxRxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_USART\\_ErrorCallback\(\)\*\*](#)

### 50.2.3 Peripheral State functions

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

- **HAL\_USART\_GetState()** API can be helpful to check in run-time the state of the USART peripheral.
- **HAL\_USART\_GetError()** API can be helpful to check in run-time the Error Code of the USART peripheral.
- **USART\_SetConfig()** API is used to set the USART communication parameters.
- **USART\_CheckIdleState()** API ensures that TEACK and/or REACK bits are set after initialization

This section contains the following APIs:

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

## 50.2.4 Detailed description of functions

### HAL\_USART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * huart)</b>
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_USART\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * huart)</b>
Function Description	DeInitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_USART\_MspInit

Function Name	<b>void HAL_USART_MspInit (USART_HandleTypeDef * huart)</b>
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_USART\_MspDeInit

Function Name	<b>void HAL_USART_MspDeInit (USART_HandleTypeDef * huart)</b>
Function Description	USART MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### HAL\_USART\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * huart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Simplex Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pTxData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> <li>• <b>Timeout:</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_USART\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pRxData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> <li>• <b>Timeout:</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_USART\_TransmitReceive**

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pTxData:</b> pointer to TX data buffer</li> <li>• <b>pRxData:</b> pointer to RX data buffer</li> <li>• <b>Size:</b> amount of data to be sent (same amount to be received)</li> <li>• <b>Timeout:</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_USART\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pTxData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_USART\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> usart handle</li> <li>• <b>pRxData:</b> pointer to data buffer</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li>• <b>Size:</b> amount of data to be received</li> <li>• <b>HAL:</b> status</li> </ul>
---------------	--

### HAL\_USART\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * huart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pTxData:</b> pointer to TX data buffer</li> <li>• <b>pRxData:</b> pointer to RX data buffer</li> <li>• <b>Size:</b> amount of data to be sent (same amount to be received)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_USART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * huart, uint8_t * pTxData, uint16_t Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pTxData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### HAL\_USART\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * huart, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> USART handle</li> <li>• <b>pRxData:</b> pointer to data buffer</li> <li>• <b>Size:</b> amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> <li>• The USART DMA transmit stream must be configured in order to generate the clock for the slave.</li> </ul>

### HAL\_USART\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * huart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
---------------	---

Function Description	Full-Duplex Transmit Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>husart:</b> usart handle</li> <li><b>pTxData:</b> pointer to TX data buffer</li> <li><b>pRxData:</b> pointer to RX data buffer</li> <li><b>Size:</b> amount of data to be received/sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the USART parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

### HAL\_USART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### HAL\_USART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### HAL\_USART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### HAL\_USART\_IRQHandler

Function Name	<b>void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)</b>
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>husart:</b> USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### HAL\_USART\_TxCpltCallback

Function Name	<b>void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)</b>
---------------	---

**husart)**

Function Description Tx Transfer completed callbacks.

Parameters • **husart:** USART handle

Return values • **None:**

**HAL\_USART\_TxHalfCpltCallback**

Function Name **void HAL\_USART\_TxHalfCpltCallback (USART\_HandleTypeDef \* husart)**

Function Description Tx Half Transfer completed callbacks.

Parameters • **husart:** USART handle

Return values • **None:**

**HAL\_USART\_RxCpltCallback**

Function Name **void HAL\_USART\_RxCpltCallback (USART\_HandleTypeDef \* husart)**

Function Description Rx Transfer completed callbacks.

Parameters • **husart:** USART handle

Return values • **None:**

**HAL\_USART\_RxHalfCpltCallback**

Function Name **void HAL\_USART\_RxHalfCpltCallback (USART\_HandleTypeDef \* husart)**

Function Description Rx Half Transfer completed callbacks.

Parameters • **husart:** USART handle

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:** USART handle

Return values • **None:**

**HAL\_USART\_ErrorCallback**

Function Name **void HAL\_USART\_ErrorCallback (USART\_HandleTypeDef \* husart)**

Function Description USART error callbacks.

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:** USART handle

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

## 50.3 USART Firmware driver defines

### 50.3.1 USART

***USART clock activation definition***

USART\_CLOCK\_DISABLE

USART\_CLOCK\_ENABLE

IS\_USART\_CLOCK

***USART clock phase definition***

USART\_PHASE\_1EDGE

USART\_PHASE\_2EDGE

IS\_USART\_PHASE

***USART polarity level definition***

USART\_POLARITY\_LOW

USART\_POLARITY\_HIGH

IS\_USART\_POLARITY

***USART error definition***

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***

<code>__HAL_USART_RESET_HANDLE_STA TE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Reset USART handle state.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>__HANDLE__</code>: specifies the USART Handle. The Handle Instance which can be USART1 or USART2.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• None</li> </ul>
<code>__HAL_USART_FLUSH_DRREGISTER</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Flush the USART Data registers.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>__HANDLE__</code>: specifies the USART Handle.</li> </ul>
<code>__HAL_USART_GET_FLAG</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Checks whether the specified USART flag is set or not.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>__HANDLE__</code>: specifies the USART Handle which can be USART1 or USART2.</li> <li>• <code>__FLAG__</code>: specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>USART_FLAG_RXACK</code>: Receive enable acknowledge flag</li> <li>– <code>USART_FLAG_TEACK</code>: Transmit enable acknowledge flag</li> <li>– <code>USART_FLAG_BUSY</code>: Busy flag</li> <li>– <code>USART_FLAG_CTS</code>: CTS Change flag</li> <li>– <code>USART_FLAG_TXE</code>: Transmit data register empty flag</li> <li>– <code>USART_FLAG_TC</code>: Transmission Complete flag</li> <li>– <code>USART_FLAG_RXNE</code>: Receive data register not empty flag</li> <li>– <code>USART_FLAG_IDLE</code>: Idle Line detection flag</li> <li>– <code>USART_FLAG_ORE</code>: OverRun Error flag</li> <li>– <code>USART_FLAG_NE</code>: Noise Error flag</li> <li>– <code>USART_FLAG_FE</code>: Framing Error flag</li> <li>– <code>USART_FLAG_PE</code>: Parity Error flag</li> </ul> </li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• The new state of <code>__FLAG__</code> (TRUE or</li> </ul>

FALSE).

#### \_HAL\_USART\_CLEAR\_FLAG

**Description:**

- Clears the specified USART pending flag.

**Parameters:**

- \_HANDLE\_: specifies the USART Handle.
- \_FLAG\_: specifies the flag to check. This parameter can be any combination of the following values:
  - USART\_CLEAR\_PEF
  - USART\_CLEAR\_FEF
  - USART\_CLEAR\_NEF
  - USART\_CLEAR\_OREF
  - USART\_CLEAR\_IDLEF
  - USART\_CLEAR\_TCF
  - USART\_CLEAR\_LBDF
  - USART\_CLEAR\_CTSF
  - USART\_CLEAR\_RTOF
  - USART\_CLEAR\_EOBF
  - USART\_CLEAR\_CMF
  - USART\_CLEAR\_WUF

**Return value:**

- None

#### \_HAL\_USART\_CLEAR\_PEFLAG

**Description:**

- Clear the USART PE pending flag.

**Parameters:**

- \_HANDLE\_: specifies the UART Handle.

**Return value:**

- None

#### \_HAL\_USART\_CLEAR\_FEFLAG

**Description:**

- Clear the USART FE pending flag.

**Parameters:**

- \_HANDLE\_: specifies the UART Handle.

**Return value:**

- None

#### \_HAL\_USART\_CLEAR\_NEFLAG

**Description:**

- Clear the UART NE pending flag.

**Parameters:**

- \_HANDLE\_: specifies the UART Handle.

**Return value:**

- None

`__HAL_USART_CLEAR_OREFLAG`**Description:**

- Clear the UART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

`__HAL_USART_CLEAR_IDLEFLAG`**Description:**

- Clear the UART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

`__HAL_USART_ENABLE_IT`**Description:**

- Enables the specified USART interrupt.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle which can be USART1 or USART2.
- `__INTERRUPT__`: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_PE: Parity Error interrupt
  - USART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

`__HAL_USART_DISABLE_IT`**Description:**

- Disables the specified USART interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle which can be USART1 or USART2.
- \_\_INTERRUPT\_\_: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_PE: Parity Error interrupt
  - USART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

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

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle which can be USART1 or USART2.
- \_\_IT\_\_: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ORE: OverRun Error interrupt
  - USART\_IT\_NE: Noise Error interrupt
  - USART\_IT\_FE: Framing Error interrupt
  - USART\_IT\_PE: Parity Error interrupt

**Return value:**

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

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

- Checks whether the specified USART interrupt source is enabled.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle which can be USART1 or USART2.
- \_\_IT\_\_: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ORE: OverRun Error interrupt
  - USART\_IT\_NE: Noise Error interrupt
  - USART\_IT\_FE: Framing Error interrupt
  - USART\_IT\_PE: Parity Error interrupt

**Return value:**

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

### \_\_HAL\_USART\_CLEAR\_IT

**Description:**

- Clears the specified USART ISR flag, in setting the proper ICR register flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle which can be USART1 or USART2.
- \_\_IT\_CLEAR\_\_: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART\_CLEAR\_PEF: Parity Error Clear Flag
  - USART\_CLEAR\_FEF: Framing Error Clear Flag
  - USART\_CLEAR\_NEF: Noise detected Clear Flag
  - USART\_CLEAR\_OREF: OverRun Error Clear Flag
  - USART\_CLEAR\_IDLEF: IDLE line detected Clear Flag
  - USART\_CLEAR\_TCF: Transmission Complete Clear Flag

- USART\_CLEAR\_CTSF: CTS Interrupt Clear Flag

**Return value:**

- None

`__HAL_USART_SEND_REQ`

**Description:**

- Set a specific USART request flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle which can be USART1 or USART2.
- `__REQ__`: specifies the request flag to set This parameter can be one of the following values:
  - USART\_RXDATA\_FLUSH\_REQUEST: Receive Data flush Request
  - USART\_TXDATA\_FLUSH\_REQUEST: Transmit data flush Request

**Return value:**

- None

`__HAL_USART_ONE_BIT_SAMPLE_ENABLE`

**Description:**

- Enables the USART one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

`__HAL_USART_ONE_BIT_SAMPLE_DISABLE`

**Description:**

- Disables the USART one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

`__HAL_USART_ENABLE`

**Description:**

- Enable USART.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. The Handle Instance which can be USART1 or USART2.

`__HAL_USART_DISABLE`**Return value:**

- None

**Description:**

- Disable USART.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

`IS_USART_BAUDRATE`**Description:**

- Check USART Baud rate.

**Parameters:**

- `BAUDRATE`: Baudrate specified by the user. The maximum Baud Rate is derived from the maximum clock on L0 (i.e. 32 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

**Return value:**

- Test: result (TRUE or FALSE).

***USART flag definitions***

<code>USART_FLAG_RXACK</code>	Receive Enable Acknowledge Flag
<code>USART_FLAG_TEACK</code>	Transmit Enable Acknowledge Flag
<code>USART_FLAG_BUSY</code>	Busy Flag
<code>USART_FLAG_CTS</code>	CTS flag
<code>USART_FLAG_CTSIF</code>	CTS interrupt flag
<code>USART_FLAG_LBDF</code>	LIN Break Detection Flag
<code>USART_FLAG_TXE</code>	Transmit Data Register Empty
<code>USART_FLAG_TC</code>	Transmission Complete
<code>USART_FLAG_RXNE</code>	Read Data Register Not Empty
<code>USART_FLAG_IDLE</code>	IDLE line detected
<code>USART_FLAG_ORE</code>	OverRun Error
<code>USART_FLAG_NE</code>	Noise detected Flag
<code>USART_FLAG_FE</code>	Framing Error
<code>USART_FLAG_PE</code>	Parity Error

***USART interruption mask definition***`USART_IT_MASK`***USART interrupt definition***

USART\_IT\_PE  
USART\_IT\_TXE  
USART\_IT\_TC  
USART\_IT\_RXNE  
USART\_IT\_IDLE  
USART\_IT\_ERR  
USART\_IT\_ORE  
USART\_IT\_NE  
USART\_IT\_FE

***USART interrupt clear flags definition***

USART\_CLEAR\_PEF      Parity Error Clear Flag  
USART\_CLEAR\_FEF      Framing Error Clear Flag  
USART\_CLEAR\_NEF      Noise detected Clear Flag  
USART\_CLEAR\_OREF      OverRun Error Clear Flag  
USART\_CLEAR\_IDLEF      IDLE line detected Clear Flag  
USART\_CLEAR\_TCF      Transmission Complete Clear Flag  
USART\_CLEAR\_CTSF      CTS Interrupt Clear Flag

***USART last bit activation definition***

USART\_LASTBIT\_DISABLE  
USART\_LASTBIT\_ENABLE  
IS\_USART\_LASTBIT

***USART mode definition***

USART\_MODE\_RX  
USART\_MODE\_TX  
USART\_MODE\_TX\_RX  
IS\_USART\_MODE

***USART parity definition***

USART\_PARITY\_NONE  
USART\_PARITY\_EVEN  
USART\_PARITY\_ODD  
IS\_USART\_PARITY

***USART request parameter definition***

USART\_RXDATA\_FLUSH\_REQUEST      Receive Data flush Request  
USART\_TXDATA\_FLUSH\_REQUEST      Transmit data flush Request  
IS\_USART\_REQUEST\_PARAMETER

***USART stop bit definition***

USART\_STOPBITS\_1

USART\_STOPBITS\_2

USART\_STOPBITS\_1\_5

IS\_USART\_STOPBITS

## 51 HAL USART Extension Driver

### 51.1 USARTEx Firmware driver defines

#### 51.1.1 USARTEx

##### ***USARTEx Exported Macros***

`USART_GETCLOCKSOURCE`

##### **Description:**

- Reports the USART clock source.

##### **Parameters:**

- `__HANDLE__`: specifies the USART Handle
- `__CLOCKSOURCE__`: output variable

##### **Return value:**

- the: USART clocking source, written in `__CLOCKSOURCE__`.

`USART_MASK_COMPUTATION`

##### **Description:**

- Reports the USART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

##### **Parameters:**

- `__HANDLE__`: specifies the USART Handle

##### **Return value:**

- mask: to apply to USART RDR register value.

##### ***Word length definition***

`USART_WORDLENGTH_7B`

`USART_WORDLENGTH_8B`

`USART_WORDLENGTH_9B`

`IS_USART_WORD_LENGTH`

## 52 HAL WWDG Generic Driver

### 52.1 WWDG Firmware driver registers structures

#### 52.1.1 WWDG\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Window*
- *uint32\_t Counter*

##### Field Documentation

- ***uint32\_t WWDG\_InitTypeDef::Prescaler***  
Specifies the prescaler value of the WWDG. This parameter can be a value of [WWDG\\_Prescaler](#)
- ***uint32\_t WWDG\_InitTypeDef::Window***  
Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number lower than Max\_Data = 0x80
- ***uint32\_t WWDG\_InitTypeDef::Counter***  
Specifies the WWDG free-running downcounter value. This parameter must be a number between Min\_Data = 0x40 and Max\_Data = 0x7F

#### 52.1.2 WWDG\_HandleTypeDefDef

##### Data Fields

- *WWDG\_TypeDef \* Instance*
- *WWDG\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_WWDG\_StateTypeDef State*

##### Field Documentation

- ***WWDG\_TypeDef\* WWDG\_HandleTypeDefDef::Instance***  
Register base address
- ***WWDG\_InitTypeDef WWDG\_HandleTypeDefDef::Init***  
WWDG required parameters
- ***HAL\_LockTypeDef WWDG\_HandleTypeDefDef::Lock***  
WWDG locking object
- ***\_\_IO HAL\_WWDG\_StateTypeDef WWDG\_HandleTypeDefDef::State***  
WWDG communication state

## 52.2 WWDG Firmware driver API description

### 52.2.1 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 counter clock = PCLK1 / Prescaler WWDG timeout = (WWDG counter clock)  
\* (counter value)
- Min-max timeout value @32 MHz(PCLK1): ~128.0 us / ~65.54 ms

### 52.2.2 How to use this driver

- Enable WWDG APB1 clock using \_\_HAL\_RCC\_WWDG\_CLK\_ENABLE().
- Set the WWDG prescaler, refresh window and counter value using HAL\_WWDG\_Init() function.
- Start the WWDG using HAL\_WWDG\_Start() function. When the WWDG is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWDG using HAL\_WWDG\_Start\_IT(). Once enabled, EWI interrupt cannot be disabled except by a system reset.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_WWDG\_Refresh() function. This operation must occur only when the counter is lower than the refresh window value already programmed.

### WWDG HAL driver macros list

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

- \_\_HAL\_WWDG\_ENABLE: Enable the WWDG peripheral
- \_\_HAL\_WWDG\_GET\_FLAG: Get the selected WWDG's flag status
- \_\_HAL\_WWDG\_CLEAR\_FLAG: Clear the WWDG's pending flags
- \_\_HAL\_WWDG\_ENABLE\_IT: Enables the WWDG early wakeup interrupt

### 52.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the WWDG\_InitTypeDef and create the associated handle
- DeInitialize the WWDG peripheral
- Initialize the WWDG MSP
- DeInitialize the WWDG MSP

This section contains the following APIs:

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

## 52.2.4 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- Handle WWDG interrupt request.

This section contains the following APIs:

- [\*HAL\\_WWDG\\_Start\(\)\*](#)
- [\*HAL\\_WWDG\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_WWDG\\_Refresh\(\)\*](#)
- [\*HAL\\_WWDG\\_IRQHandler\(\)\*](#)
- [\*HAL\\_WWDG\\_WakeupCallback\(\)\*](#)

## 52.2.5 Peripheral State functions

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

This section contains the following APIs:

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

## 52.2.6 Detailed description of functions

### **HAL\_WWDG\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwdg)</b>
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

### **HAL\_WWDG\_DelInit**

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

**HAL\_WWDG\_MspInit**

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

**HAL\_WWDG\_MspDeInit**

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

**HAL\_WWDG\_WakeupCallback**

Function Name	<b>void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> : pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**HAL\_WWDG\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Starts the WWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> : pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL:</b> status</li> </ul>

**HAL\_WWDG\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Starts the WWDG with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> : pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.</li> </ul>

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

### HAL\_WWDG\_Refresh

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwdg, uint32_t Counter)</b>
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> <li><b>hwdg:</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> <li><b>Counter:</b> value of counter to put in WWDG counter</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> status</li> </ul>

### HAL\_WWDG\_IRQHandler

Function Name	<b>void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwdg)</b>
Function Description	Handles WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hwdg:</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled using __HAL_WWDG_ENABLE_IT() macro. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.</li> </ul>

### HAL\_WWDG\_GetState

Function Name	<b>HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwdg)</b>
Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> <li><b>hwdg:</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL:</b> state</li> </ul>

## 52.3 WWDG Firmware driver defines

### 52.3.1 WWDG

**WWDG BitAddress AliasRegion**

**WWDG\_CFR\_BASE**

**WWDG Exported Constants**



IS\_WWDG\_PRESCALER

IS\_WWDG\_WINDOW

IS\_WWDG\_COUNTER

**WWDG Exported Macros**

`_HAL_WWDG_RESET_HANDLE_STATE` **Description:**

- Reset WWDG handle state.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None

`_HAL_WWDG_ENABLE`

**Description:**

- Enables the WWDG peripheral.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None

`_HAL_WWDG_DISABLE`

**Description:**

- Disables the WWDG peripheral.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None

**Notes:**

- WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.

`_HAL_WWDG_ENABLE_IT`

**Description:**

- Enables the WWDG early wakeup interrupt.

**Parameters:**

- `_HANDLE_`: WWDG handle
- `_INTERRUPT_`: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG\_IT\_EWI: Early wakeup interrupt

**Return value:**

- None

**Notes:**

- Once enabled this interrupt cannot be disabled except by a system reset.

`_HAL_WWDG_DISABLE_IT`**Description:**

- Disables the WWDG early wakeup interrupt.

**Parameters:**

- `_HANDLE_`: WWDG handle;
- `_INTERRUPT_`: specifies the interrupt to disable.
  - `WWDG_IT_EWI`: Early wakeup interrupt

**Return value:**

- None

**Notes:**

- WARNING: This is a dummy macro for HAL code alignment. Once enabled this interrupt cannot be disabled except by a system reset.

`_HAL_WWDG_GET_IT`**Description:**

- Gets the selected WWDG's it status.

**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 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:**

- Gets the selected WWDG's flag status.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - WWDG\_FLAG\_EWIF: Early wakeup interrupt flag

**Return value:**

- The new state of WWDG\_FLAG (SET or RESET).

\_\_HAL\_WWDG\_CLEAR\_FLAG

**Description:**

- Clears the WWDG's pending flags.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one of the following values:
  - WWDG\_FLAG\_EWIF: Early wakeup interrupt flag

**Return value:**

- None

\_\_HAL\_WWDG\_GET\_IT\_SOURCE

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: WWDG Handle.
- \_\_INTERRUPT\_\_: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - WWDG\_IT\_EWI: Early Wakeup Interrupt

**Return value:**

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

**WWDG Flag definition**

WWDG\_FLAG\_EWIF Early wakeup interrupt flag

**WWDG Interrupt definition**

WWDG\_IT\_EWI

**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

## 53 LL ADC Generic Driver

### 53.1 ADC Firmware driver registers structures

#### 53.1.1 LL\_ADC\_CommonInitTypeDef

##### Data Fields

- *uint32\_t CommonClock*

##### Field Documentation

- *uint32\_t LL\_ADC\_CommonInitTypeDef::CommonClock*

Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of [ADC\\_LL\\_EC\\_COMMON\\_CLOCK\\_SOURCE](#)This feature can be modified afterwards using unitary function [LL\\_ADC\\_SetCommonClock\(\)](#).

#### 53.1.2 LL\_ADC\_InitTypeDef

##### Data Fields

- *uint32\_t Clock*
- *uint32\_t Resolution*
- *uint32\_t DataAlignment*
- *uint32\_t LowPowerMode*

##### Field Documentation

- *uint32\_t LL\_ADC\_InitTypeDef::Clock*

Set ADC instance clock source and prescaler. This parameter can be a value of [ADC\\_LL\\_EC\\_CLOCK\\_SOURCE](#)

**Note:**On this STM32 serie, this parameter has some clock ratio constraints: ADC clock synchronous (from PCLK) with prescaler 1 must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). This feature can be modified afterwards using unitary function [LL\\_ADC\\_SetClock\(\)](#). For more details, refer to description of this function.

- *uint32\_t LL\_ADC\_InitTypeDef::Resolution*

Set ADC resolution. This parameter can be a value of [ADC\\_LL\\_EC\\_RESOLUTION](#)This feature can be modified afterwards using unitary function [LL\\_ADC\\_SetResolution\(\)](#).

- *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::LowPowerMode*

Set ADC low power mode. This parameter can be a value of

**ADC\_LL\_EC\_LP\_MODE**This feature can be modified afterwards using unitary function **LL\_ADC\_SetLowPowerMode()**.

### 53.1.3 LL\_ADC\_REG\_InitTypeDef

#### Data Fields

- *uint32\_t TriggerSource*
- *uint32\_t SequencerDiscont*
- *uint32\_t ContinuousMode*
- *uint32\_t DMATransfer*
- *uint32\_t Overrun*

#### 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, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL\_ADC\_REG\_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL\_ADC\_REG\_SetTriggerSource()**.

- *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 (several ADC channels enabled in group regular sequencer). 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()**.

- *uint32\_t LL\_ADC\_REG\_InitTypeDef::Overrun*

Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of **ADC\_LL\_EC\_REG\_OVR\_DATA\_BEHAVIOR**This feature can be modified afterwards using unitary function

**LL\_ADC\_REG\_SetOverrun()**.

## 53.2 ADC Firmware driver API description

### 53.2.1 Detailed description of functions

#### LL\_ADC\_DMA\_GetRegAddr

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr(ADC_TypeDef * ADCx, uint32_t Register)</code>
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	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Register:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_ADC_REG_REGULAR_DATA</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>ADC:</b> register address</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.</li> <li>• This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example:  <code>LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1,  LL_ADC_DMA_GetRegAddr(ADC1,  LL_ADC_REG_REGULAR_DATA), (uint32_t)&amp;&lt; array  or variable &gt;,  LL_DMA_DIRECTION_PERIPH_TO_MEMORY);</code> </li> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DATA <code>LL_ADC_DMA_GetRegAddr</code></li> </ul>

#### LL\_ADC\_SetCommonClock

Function Name	<code>__STATIC_INLINE void LL_ADC_SetCommonClock(ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)</code>
Function Description	Set parameter common to several ADC: Clock source and prescaler.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCxy_COMMON:</b> ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)</li> <li>• <b>CommonClock:</b> This parameter can be one of the following values: (1) ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous. (refer to function <code>LL_ADC_SetClock()</code> ).  <ul style="list-style-type: none"> <li>- <code>LL_ADC_CLOCK_ASYNC_DIV1</code> (1)</li> <li>- <code>LL_ADC_CLOCK_ASYNC_DIV2</code> (1)</li> <li>- <code>LL_ADC_CLOCK_ASYNC_DIV4</code> (1)</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_ADC_CLOCK_ASYNC_DIV6 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV8 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV10 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV12 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV16 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV32 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV64 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV128 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV256 (1)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function <code>LL_ADC_IsEnabled()</code> for each ADC instance or by using helper macro helper macro <code>__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE()</code>.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PRESC <code>LL_ADC_SetCommonClock</code></li> </ul>

### LL\_ADC\_GetCommonClock

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetCommonClock(ADC_Common_TypeDef * ADCxy_COMMON)</code>
Function Description	Get parameter common to several ADC: Clock source and prescaler.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCxy_COMMON:</b> ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: (1) ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous. (refer to function <code>LL_ADC_SetClock()</code> ).</li> </ul> <ul style="list-style-type: none"> <li>- LL_ADC_CLOCK_ASYNC_DIV1 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV2 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV4 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV6 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV8 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV10 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV12 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV16 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV32 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV64 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV128 (1)</li> <li>- LL_ADC_CLOCK_ASYNC_DIV256 (1)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PRESC <code>LL_ADC_GetCommonClock</code></li> </ul>

**LL\_ADC\_SetCommonFrequencyMode**

Function Name	<code>__STATIC_INLINE void LL_ADC_SetCommonFrequencyMode(ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Resolution)</code>
Function Description	Set parameter common to several ADC: Clock low frequency mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCxy_COMMON:</b> ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)</li> <li>• <b>Resolution:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_ADC_CLOCK_FREQ_MODE_HIGH</code></li> <li>- <code>LL_ADC_CLOCK_FREQ_MODE_LOW</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR LFMEN LL_ADC_SetCommonFrequencyMode</li> </ul>

**LL\_ADC\_GetCommonFrequencyMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetCommonFrequencyMode(ADC_Common_TypeDef * ADCxy_COMMON)</code>
Function Description	Get parameter common to several ADC: Clock low frequency mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCxy_COMMON:</b> ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_ADC_CLOCK_FREQ_MODE_HIGH</code></li> <li>- <code>LL_ADC_CLOCK_FREQ_MODE_LOW</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR LFMEN LL_ADC_GetCommonFrequencyMode</li> </ul>

**LL\_ADC\_SetCommonPathInternalCh**

Function Name	<code>__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh(ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)</code>
Function Description	Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCxy_COMMON:</b> ADC common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_ADC_COMMON_INSTANCE()</code>)</li> </ul>

	<ul style="list-style-type: none"> <li><b>PathInternal:</b> This parameter can be a combination of the following values: (*) value not defined in all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx.           <ul style="list-style-type: none"> <li>- LL_ADC_PATH_INTERNAL_NONE</li> <li>- LL_ADC_PATH_INTERNAL_VREFINT</li> <li>- LL_ADC_PATH_INTERNAL_TEMPSENSOR</li> <li>- LL_ADC_PATH_INTERNAL_VLCD (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT   LL_ADC_PATH_INTERNAL_TEMPSENSOR)</li> <li>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_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US.</li> <li>ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.</li> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR VREFEN LL_ADC_SetCommonPathInternalCh</li> <li>CCR TSEN LL_ADC_SetCommonPathInternalCh</li> <li>CCR VLCDEN LL_ADC_SetCommonPathInternalCh</li> </ul>

### LL\_ADC\_GetCommonPathInternalCh

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)</code>
Function Description	Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).
Parameters	<ul style="list-style-type: none"> <li><b>ADCxy_COMMON:</b> ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE() )</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be a combination of the following values: (*) value not defined in all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx.           <ul style="list-style-type: none"> <li>- LL_ADC_PATH_INTERNAL_NONE</li> <li>- LL_ADC_PATH_INTERNAL_VREFINT</li> <li>- LL_ADC_PATH_INTERNAL_TEMPSENSOR</li> <li>- LL_ADC_PATH_INTERNAL_VLCD (*)</li> </ul> </li> </ul>

Notes	<ul style="list-style-type: none"> <li>One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT   LL_ADC_PATH_INTERNAL_TEMPSENSOR)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR VREFEN LL_ADC_GetCommonPathInternalCh</li> <li>CCR TSEN LL_ADC_GetCommonPathInternalCh</li> <li>CCR VLCDEN LL_ADC_GetCommonPathInternalCh</li> </ul>

## LL\_ADC\_SetClock

Function Name	<code>__STATIC_INLINE void LL_ADC_SetClock (ADC_TypeDef * ADCx, uint32_t ClockSource)</code>
Function Description	Set ADC instance clock source and prescaler.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> <li><b>ClockSource:</b> This parameter can be one of the following values: (1) Asynchronous clock prescaler can be configured using function LL_ADC_SetCommonClock(). <ul style="list-style-type: none"> <li>– LL_ADC_CLOCK_SYNC_PCLK_DIV4</li> <li>– LL_ADC_CLOCK_SYNC_PCLK_DIV2</li> <li>– LL_ADC_CLOCK_SYNC_PCLK_DIV1 (2)</li> <li>– LL_ADC_CLOCK_ASYNC (1)</li> </ul> </li> <li>(2) Caution: This parameter has some clock ratio constraints: This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). Refer to reference manual.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR2 CKMODE LL_ADC_SetClock</li> </ul>

## LL\_ADC\_GetClock

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetClock (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC instance clock source and prescaler.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: (1) Asynchronous clock prescaler can be retrieved using function LL_ADC_GetCommonClock(). <ul style="list-style-type: none"> <li>– LL_ADC_CLOCK_SYNC_PCLK_DIV4</li> <li>– LL_ADC_CLOCK_SYNC_PCLK_DIV2</li> <li>– LL_ADC_CLOCK_SYNC_PCLK_DIV1 (2)</li> <li>– LL_ADC_CLOCK_ASYNC (1)</li> </ul> </li> <li>(2) Caution: This parameter has some clock ratio constraints: This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty</li> </ul>

cycle). Refer to reference manual.

Reference Manual to  
LL API cross  
reference:

- CFGR2 CKMODE LL\_ADC\_GetClock

### **LL\_ADC\_SetCalibrationFactor**

Function Name	<b><code>_STATIC_INLINE void LL_ADC_SetCalibrationFactor(ADC_TypeDef * ADCx, uint32_t CalibrationFactor)</code></b>
Function Description	Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>CalibrationFactor:</b> Value between Min_Data=0x00 and Max_Data=0x7F</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function is intended to set calibration parameters without having to perform a new calibration using <code>LL_ADC_StartCalibration()</code>.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALFACT CALFACT LL_ADC_SetCalibrationFactor</li> </ul>

### **LL\_ADC\_GetCalibrationFactor**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor(ADC_TypeDef * ADCx)</code></b>
Function Description	Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x7F</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Calibration factors are set by hardware after performing a calibration run using function <code>LL_ADC_StartCalibration()</code>.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALFACT CALFACT LL_ADC_GetCalibrationFactor</li> </ul>

### **LL\_ADC\_SetResolution**

Function Name	<b><code>_STATIC_INLINE void LL_ADC_SetResolution(ADC_TypeDef * ADCx, uint32_t Resolution)</code></b>
Function Description	Set ADC resolution.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Resolution:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>LL_ADC_RESOLUTION_12B</code></li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_ADC_RESOLUTION_10B</li> <li>- LL_ADC_RESOLUTION_8B</li> <li>- LL_ADC_RESOLUTION_6B</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 RES LL_ADC_SetResolution</li> </ul>

### LL\_ADC\_GetResolution

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetResolution(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC resolution.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_ADC_RESOLUTION_12B</li> <li>- LL_ADC_RESOLUTION_10B</li> <li>- LL_ADC_RESOLUTION_8B</li> <li>- LL_ADC_RESOLUTION_6B</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 RES LL_ADC_GetResolution</li> </ul>

### LL\_ADC\_SetDataAlignment

Function Name	<code>__STATIC_INLINE void LL_ADC_SetDataAlignment(ADC_TypeDef * ADCx, uint32_t DataAlignment)</code>
Function Description	Set ADC conversion data alignment.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>DataAlignment:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_ADC_DATA_ALIGN_RIGHT</li> <li>- LL_ADC_DATA_ALIGN_LEFT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Refer to reference manual for alignments formats dependencies to ADC resolutions.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 ALIGN LL_ADC_SetDataAlignment</li> </ul>

**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:

- CFGR1 ALIGN LL\_ADC\_GetDataAlignment

**LL\_ADC\_SetLowPowerMode**

Function Name      **STATIC\_INLINE void LL\_ADC\_SetLowPowerMode(ADC\_TypeDef \* ADCx, uint32\_t LowPowerMode)**

Function Description      Set ADC low power mode.

Parameters      • **ADCx:** ADC instance  
• **LowPowerMode:** This parameter can be one of the following values:  
– LL\_ADC\_LP\_MODE\_NONE  
– LL\_ADC\_LP\_AUTOWAIT  
– LL\_ADC\_LP\_AUTOPOWEROFF  
– LL\_ADC\_LP\_AUTOWAIT\_AUTOPOWEROFF

Return values      • **None:**

Notes      • Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL\_ADC\_LP\_MODE\_AUTOOFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup

time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".

- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to  
LL API cross  
reference:

- CFGR1 WAIT LL\_ADC\_SetLowPowerMode
- CFGR1 AUTOFF LL\_ADC\_SetLowPowerMode

### **LL\_ADC\_GetLowPowerMode**

Function Name	<b><code>STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode(ADC_TypeDef * ADCx)</code></b>
Function Description	Get ADC low power mode:
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_LP_MODE_NONE</li> <li>- LL_ADC_LP_AUTOWAIT</li> <li>- LL_ADC_LP_AUTOPOWEROFF</li> <li>- LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_MODE_AUTOFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".</li> <li>With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start,</li> </ul>

independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to  
LL API cross  
reference:

- CFGR1 WAIT LL\_ADC\_GetLowPowerMode
- CFGR1 AUTOFF LL\_ADC\_GetLowPowerMode

### **LL\_ADC\_SetSamplingTimeCommonChannels**

Function Name	<b>STATIC_INLINE void LL_ADC_SetSamplingTimeCommonChannels (ADC_TypeDef * ADCx, uint32_t SamplingTime)</b>
Function Description	Set sampling time common to a group of channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>SamplingTime:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_SAMPLINGTIME_1CYCLE_5</li> <li>– LL_ADC_SAMPLINGTIME_7CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_13CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_28CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_41CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_55CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_71CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_239CYCLES_5</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Unit: ADC clock cycles.</li> <li>• On this STM32 serie, sampling time scope is on ADC instance: Sampling time common to all channels. (on some other STM32 families, sampling time is channel wise)</li> <li>• 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, ...).</li> <li>• Conversion time is the addition of sampling time and processing time. On this STM32 serie, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits</li> <li>• In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SMPR SMP LL_ADC_SetSamplingTimeCommonChannels</li> </ul>

**LL\_ADC\_GetSamplingTimeCommonChannels**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetSamplingTimeCommonChannels (ADC_TypeDef * ADCx)</code>
Function Description	Get sampling time common to a group of channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_SAMPLINGTIME_1CYCLE_5</li> <li>– LL_ADC_SAMPLINGTIME_7CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_13CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_28CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_41CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_55CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_71CYCLES_5</li> <li>– LL_ADC_SAMPLINGTIME_239CYCLES_5</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Unit: ADC clock cycles.</li> <li>• On this STM32 serie, sampling time scope is on ADC instance: Sampling time common to all channels. (on some other STM32 families, sampling time is channel wise)</li> <li>• Conversion time is the addition of sampling time and processing time. Refer to reference manual for ADC processing time of this STM32 serie.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SMPR SMP LL_ADC_GetSamplingTimeCommonChannels</li> </ul>

**LL\_ADC\_REG\_SetTriggerSource**

Function Name	<code>__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)</code>
Function Description	Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>TriggerSource:</b> This parameter can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>– LL_ADC_REG_TRIG_SOFTWARE</li> <li>– LL_ADC_REG_TRIG_EXT_TIM6_TRGO</li> <li>– LL_ADC_REG_TRIG_EXT_TIM21_CH2</li> <li>– LL_ADC_REG_TRIG_EXT_TIM2_TRGO</li> <li>– LL_ADC_REG_TRIG_EXT_TIM2_CH4</li> <li>– LL_ADC_REG_TRIG_EXT_TIM22_TRGO</li> <li>– LL_ADC_REG_TRIG_EXT_TIM2_CH3 (*)</li> <li>– LL_ADC_REG_TRIG_EXT_TIM3_TRGO</li> <li>– LL_ADC_REG_TRIG_EXT EXTI_LINE11</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to</li> </ul>

- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
  - On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- Reference Manual to LL API cross reference:
- CFGR1 EXTSEL LL\_ADC\_REG\_SetTriggerSource
  - CFGR1 EXTEN LL\_ADC\_REG\_SetTriggerSource

### **LL\_ADC\_REG\_GetTriggerSource**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line).
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_ADC_REG_TRIG_SOFTWARE</li> <li>- LL_ADC_REG_TRIG_EXT_TIM6_TRGO</li> <li>- LL_ADC_REG_TRIG_EXT_TIM21_CH2</li> <li>- LL_ADC_REG_TRIG_EXT_TIM2_TRGO</li> <li>- LL_ADC_REG_TRIG_EXT_TIM2_CH4</li> <li>- LL_ADC_REG_TRIG_EXT_TIM22_TRGO</li> <li>- LL_ADC_REG_TRIG_EXT_TIM2_CH3 (*)</li> <li>- LL_ADC_REG_TRIG_EXT_TIM3_TRGO</li> <li>- LL_ADC_REG_TRIG_EXT EXTI_LINE11</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Availability of parameters of trigger sources from timer depends on timers availability on the selected device.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 EXTSEL LL_ADC_REG_GetTriggerSource</li> <li>• CFGR1 EXTEN LL_ADC_REG_GetTriggerSource</li> </ul>

### **LL\_ADC\_REG\_IsTriggerSourceSWStart**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion trigger source internal (SW start) or external.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li>-: 0 trigger source external trigger</li> <li>- 1 trigger source SW start.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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().</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR1 EXTEN LL_ADC_REG_IsTriggerSourceSWStart</li> </ul>

### LL\_ADC\_REG\_SetTriggerEdge

Function Name	<b><code>__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge(ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)</code></b>
Function Description	Set ADC group regular conversion trigger polarity.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> <li><b>ExternalTriggerEdge:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_REG_TRIG_EXT_RISING</li> <li>- LL_ADC_REG_TRIG_EXT_FALLING</li> <li>- LL_ADC_REG_TRIG_EXT_RISINGFALLING</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Applicable only for trigger source set to external trigger.</li> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR1 EXTEN LL_ADC_REG_SetTriggerEdge</li> </ul>

### LL\_ADC\_REG\_GetTriggerEdge

Function Name	<b><code>__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge(ADC_TypeDef * ADCx)</code></b>
Function Description	Get ADC group regular conversion trigger polarity.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_REG_TRIG_EXT_RISING</li> <li>- LL_ADC_REG_TRIG_EXT_FALLING</li> <li>- LL_ADC_REG_TRIG_EXT_RISINGFALLING</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>Applicable only for trigger source set to external trigger.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR1 EXTEN LL_ADC_REG_SetTriggerEdge</li> </ul>

### LL\_ADC\_REG\_SetSequencerScanDirection

Function Name	<b><code>__STATIC_INLINE void LL_ADC_REG_SetSequencerScanDirection (ADC_TypeDef *</code></b>
---------------	--

**ADCx, uint32\_t ScanDirection)**

Function Description	Set ADC group regular sequencer scan direction.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>ScanDirection:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_SEQ_SCAN_DIR_FORWARD</li> <li>– LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On some other STM32 families, this setting is not available and the default scan direction is forward.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 SCANDIR</li> <li>  LL_ADC_REG_SetSequencerScanDirection</li> </ul>

**LL\_ADC\_REG\_GetSequencerScanDirection**

Function Name	<b>_STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerScanDirection (ADC_TypeDef * ADCx)</b>
Function Description	Get ADC group regular sequencer scan direction.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_SEQ_SCAN_DIR_FORWARD</li> <li>– LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On some other STM32 families, this setting is not available and the default scan direction is forward.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 SCANDIR</li> <li>  LL_ADC_REG_SetSequencerScanDirection</li> </ul>

**LL\_ADC\_REG\_SetSequencerDiscont**

Function Name	<b>_STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)</b>
Function Description	Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>SeqDiscont:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_SEQ_DISCONT_DISABLE</li> <li>– LL_ADC_REG_SEQ_DISCONT_1RANK</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Notes	<ul style="list-style-type: none"> <li>It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.</li> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR1 DISCEN LL_ADC_REG_SetSequencerDiscont</li> <li>•</li> </ul>

### LL\_ADC\_REG\_GetSequencerDiscont

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_REG_SEQ_DISCONT_DISABLE</li> <li>- LL_ADC_REG_SEQ_DISCONT_1RANK</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR1 DISCEN LL_ADC_REG_SetSequencerDiscont</li> <li>•</li> </ul>

### LL\_ADC\_REG\_SetSequencerChannels

Function Name	<code>__STATIC_INLINE void LL_ADC_REG_SetSequencerChannels (ADC_TypeDef * ADCx, uint32_t Channel)</code>
Function Description	Set ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> <li><b>Channel:</b> This parameter can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. <ul style="list-style-type: none"> <li>- LL_ADC_CHANNEL_0</li> <li>- LL_ADC_CHANNEL_1</li> <li>- LL_ADC_CHANNEL_2</li> <li>- LL_ADC_CHANNEL_3</li> <li>- LL_ADC_CHANNEL_4</li> <li>- LL_ADC_CHANNEL_5</li> <li>- LL_ADC_CHANNEL_6</li> <li>- LL_ADC_CHANNEL_7</li> <li>- LL_ADC_CHANNEL_8</li> <li>- LL_ADC_CHANNEL_9</li> <li>- LL_ADC_CHANNEL_10</li> <li>- LL_ADC_CHANNEL_11</li> <li>- LL_ADC_CHANNEL_12</li> <li>- LL_ADC_CHANNEL_13</li> <li>- LL_ADC_CHANNEL_14</li> </ul> </li> </ul>

- LL\_ADC\_CHANNEL\_15
- LL\_ADC\_CHANNEL\_16 (1)
- LL\_ADC\_CHANNEL\_17
- LL\_ADC\_CHANNEL\_18
- LL\_ADC\_CHANNEL\_VREFINT
- LL\_ADC\_CHANNEL\_TEMPSENSOR
- LL\_ADC\_CHANNEL\_VLCD (1)

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Set channels selected by overwriting the current sequencer configuration.
- On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.
- 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().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example:  
(LL\_ADC\_CHANNEL\_4 | LL\_ADC\_CHANNEL\_12 | ...)

Reference Manual to  
LL API cross  
reference:

- CHSELR CHSEL0 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL1 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL2 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL3 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL4 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL5 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL6 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL7 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL8 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL9 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL10 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL11 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL12 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL13 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL14 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL15 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL16 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL17 LL\_ADC\_REG\_SetSequencerChannels
- CHSELR CHSEL18 LL\_ADC\_REG\_SetSequencerChannels

**LL\_ADC\_REG\_SetSequencerChAdd**

Function Name

**\_STATIC\_INLINE void LL\_ADC\_REG\_SetSequencerChAdd**

**(ADC\_TypeDef \* ADCx, uint32\_t Channel)**

Function Description	Add channel to ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Channel:</b> This parameter can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. <ul style="list-style-type: none"> <li>- LL_ADC_CHANNEL_0</li> <li>- LL_ADC_CHANNEL_1</li> <li>- LL_ADC_CHANNEL_2</li> <li>- LL_ADC_CHANNEL_3</li> <li>- LL_ADC_CHANNEL_4</li> <li>- LL_ADC_CHANNEL_5</li> <li>- LL_ADC_CHANNEL_6</li> <li>- LL_ADC_CHANNEL_7</li> <li>- LL_ADC_CHANNEL_8</li> <li>- LL_ADC_CHANNEL_9</li> <li>- LL_ADC_CHANNEL_10</li> <li>- LL_ADC_CHANNEL_11</li> <li>- LL_ADC_CHANNEL_12</li> <li>- LL_ADC_CHANNEL_13</li> <li>- LL_ADC_CHANNEL_14</li> <li>- LL_ADC_CHANNEL_15</li> <li>- LL_ADC_CHANNEL_16 (1)</li> <li>- LL_ADC_CHANNEL_17</li> <li>- LL_ADC_CHANNEL_18</li> <li>- LL_ADC_CHANNEL_VREFINT</li> <li>- LL_ADC_CHANNEL_TEMPSENSOR</li> <li>- LL_ADC_CHANNEL_VLCD (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Set channels selected by adding them to the current sequencer configuration.</li> <li>• On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.</li> <li>• Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.</li> <li>• 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().</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> <li>• One or several values can be selected. Example: (LL_ADC_CHANNEL_4   LL_ADC_CHANNEL_12   ...)</li> </ul>

- Reference Manual to LL API cross reference:
- CHSELR CHSEL0 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL1 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL2 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL3 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL4 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL5 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL6 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL7 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL8 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL9 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL10 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL11 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL12 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL13 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL14 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL15 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL16 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL17 LL\_ADC\_REG\_SetSequencerChAdd
  - CHSELR CHSEL18 LL\_ADC\_REG\_SetSequencerChAdd

### LL\_ADC\_REG\_SetSequencerChRem

Function Name	<code>__STATIC_INLINE void LL_ADC_REG_SetSequencerChRem(ADC_TypeDef * ADCx, uint32_t Channel)</code>
Function Description	Remove channel to ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Channel:</b> This parameter can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx. <ul style="list-style-type: none"> <li>- LL_ADC_CHANNEL_0</li> <li>- LL_ADC_CHANNEL_1</li> <li>- LL_ADC_CHANNEL_2</li> <li>- LL_ADC_CHANNEL_3</li> <li>- LL_ADC_CHANNEL_4</li> <li>- LL_ADC_CHANNEL_5</li> <li>- LL_ADC_CHANNEL_6</li> <li>- LL_ADC_CHANNEL_7</li> <li>- LL_ADC_CHANNEL_8</li> <li>- LL_ADC_CHANNEL_9</li> <li>- LL_ADC_CHANNEL_10</li> <li>- LL_ADC_CHANNEL_11</li> <li>- LL_ADC_CHANNEL_12</li> <li>- LL_ADC_CHANNEL_13</li> <li>- LL_ADC_CHANNEL_14</li> <li>- LL_ADC_CHANNEL_15</li> <li>- LL_ADC_CHANNEL_16 (1)</li> <li>- LL_ADC_CHANNEL_17</li> <li>- LL_ADC_CHANNEL_18</li> <li>- LL_ADC_CHANNEL_VREFINT</li> <li>- LL_ADC_CHANNEL_TEMPSENSOR</li> </ul> </li> </ul>

	– LL_ADC_CHANNEL_VLCD (1)
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Set channels selected by removing them to the current sequencer configuration.</li> <li>On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.</li> <li>Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.</li> <li>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().</li> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> <li>One or several values can be selected. Example: (LL_ADC_CHANNEL_4   LL_ADC_CHANNEL_12   ...)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CHSEL0 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL1 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL2 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL3 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL4 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL5 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL6 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL7 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL8 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL9 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL10 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL11 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL12 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL13 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL14 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL15 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL16 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL17 LL_ADC_REG_SetSequencerChRem</li> <li>CHSEL18 LL_ADC_REG_SetSequencerChRem</li> </ul>

### LL\_ADC\_REG\_GetSequencerChannels

Function Name	<b>STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerChannels (ADC_TypeDef * ADCx)</b>
Function Description	Get ADC group regular sequence: channel on rank corresponding to channel number.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be a combination of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx.           <ul style="list-style-type: none"> <li>- LL_ADC_CHANNEL_0</li> <li>- LL_ADC_CHANNEL_1</li> <li>- LL_ADC_CHANNEL_2</li> <li>- LL_ADC_CHANNEL_3</li> <li>- LL_ADC_CHANNEL_4</li> <li>- LL_ADC_CHANNEL_5</li> <li>- LL_ADC_CHANNEL_6</li> <li>- LL_ADC_CHANNEL_7</li> <li>- LL_ADC_CHANNEL_8</li> <li>- LL_ADC_CHANNEL_9</li> <li>- LL_ADC_CHANNEL_10</li> <li>- LL_ADC_CHANNEL_11</li> <li>- LL_ADC_CHANNEL_12</li> <li>- LL_ADC_CHANNEL_13</li> <li>- LL_ADC_CHANNEL_14</li> <li>- LL_ADC_CHANNEL_15</li> <li>- LL_ADC_CHANNEL_16 (1)</li> <li>- LL_ADC_CHANNEL_17</li> <li>- LL_ADC_CHANNEL_18</li> <li>- LL_ADC_CHANNEL_VREFINT</li> <li>- LL_ADC_CHANNEL_TEMPSENSOR</li> <li>- LL_ADC_CHANNEL_VLCD (1)</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function performs: Channels order reading into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).</li> <li>• On this STM32 serie, ADC group regular sequencer is not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.</li> <li>• Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.</li> <li>• 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().</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> <li>• One or several values can be retrieved. Example: (LL_ADC_CHANNEL_4   LL_ADC_CHANNEL_12   ...)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CHSELR CHSEL0 LL_ADC_REG_GetSequencerChannels</li> <li>• CHSELR CHSEL1 LL_ADC_REG_GetSequencerChannels</li> <li>• CHSELR CHSEL2 LL_ADC_REG_GetSequencerChannels</li> <li>• CHSELR CHSEL3 LL_ADC_REG_GetSequencerChannels</li> <li>• CHSELR CHSEL4 LL_ADC_REG_GetSequencerChannels</li> <li>• CHSELR CHSEL5 LL_ADC_REG_GetSequencerChannels</li> <li>• CHSELR CHSEL6 LL_ADC_REG_GetSequencerChannels</li> </ul>

- CHSELR CHSEL7 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL8 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL9 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL10 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL11 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL12 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL13 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL14 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL15 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL16 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL17 LL\_ADC\_REG\_GetSequencerChannels
- CHSELR CHSEL18 LL\_ADC\_REG\_GetSequencerChannels

### **LL\_ADC\_REG\_SetContinuousMode**

Function Name	<b><code>_STATIC_INLINE void LL_ADC_REG_SetContinuousMode(ADC_TypeDef * ADCx, uint32_t Continuous)</code></b>
Function Description	Set ADC continuous conversion mode on ADC group regular.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Continuous:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_CONV_SINGLE</li> <li>– LL_ADC_REG_CONV_CONTINUOUS</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.</li> <li>• It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 CONT LL_ADC_REG_SetContinuousMode</li> </ul>

### **LL\_ADC\_REG\_GetContinuousMode**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode(ADC_TypeDef * ADCx)</code></b>
Function Description	Get ADC continuous conversion mode on ADC group regular.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_CONV_SINGLE</li> <li>– LL_ADC_REG_CONV_CONTINUOUS</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively</li> </ul>

automatically.

Reference Manual to  
LL API cross  
reference:

- CFGR1 CONT LL\_ADC\_REG\_GetContinuousMode

### LL\_ADC\_REG\_SetDMATransfer

Function Name	<code>__STATIC_INLINE void LL_ADC_REG_SetDMATransfer(ADC_TypeDef * ADCx, uint32_t DMATransfer)</code>
Function Description	Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>DMATransfer:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_DMA_TRANSFER_NONE</li> <li>– LL_ADC_REG_DMA_TRANSFER_LIMITED</li> <li>– LL_ADC_REG_DMA_TRANSFER_UNLIMITED</li> </ul> </li> <li>• <b>None:</b></li> </ul>
Return values	
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• 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).</li> <li>• To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 DMAEN LL_ADC_REG_SetDMATransfer</li> <li>• CFGR1 DMACFG LL_ADC_REG_SetDMATransfer</li> </ul>

### LL\_ADC\_REG\_GetDMATransfer

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.
Parameters	
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_REG_DMA_TRANSFER_NONE</li> <li>– LL_ADC_REG_DMA_TRANSFER_LIMITED</li> </ul> </li> </ul>

- Notes
- LL\_ADC\_REG\_DMA\_TRANSFER\_UNLIMITED
  - 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:
- CFGR1 DMAEN LL\_ADC\_REG\_GetDMATransfer
  - CFGR1 DMACFG LL\_ADC\_REG\_GetDMATransfer

### **LL\_ADC\_REG\_SetOverrun**

Function Name	<b><code>_STATIC_INLINE void LL_ADC_REG_SetOverrun(ADC_TypeDef * ADCx, uint32_t Overrun)</code></b>
Function Description	Set ADC group regular behavior in case of overrun: data preserved or overwritten.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Overrun:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_REG_OVR_DATA_PRESERVED</li> <li>- LL_ADC_REG_OVR_DATA_OVERWRITTEN</li> </ul> </li> <li>• <b>None:</b></li> </ul>
Return values	
Notes	<ul style="list-style-type: none"> <li>• Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR1 OVRMOD LL_ADC_REG_SetOverrun</li> </ul>

### **LL\_ADC\_REG\_GetOverrun**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun(ADC_TypeDef * ADCx)</code></b>
Function Description	Get ADC group regular behavior in case of overrun: data preserved or overwritten.

- |   |  |
|---|--|
| Parameters  | <ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>  |
| Return values                                     | <ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_ADC_REG_OVR_DATA_PRESERVED</li> <li>- LL_ADC_REG_OVR_DATA_OVERWRITTEN</li> </ul> </li> </ul> |
| Reference Manual to<br>LL API cross<br>reference: | <ul style="list-style-type: none"> <li>• CFGR1 OVRMOD LL_ADC_REG_GetOverrun</li> </ul>   |

### **LL\_ADC\_SetAnalogWDMonitChannels**

Function Name	<b><code>__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels(ADC_TypeDef * ADCx, uint32_t AWDChannelGroup)</code></b>
Function Description	Set ADC analog watchdog monitored channels: a single channel or all channels, on ADC group regular.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>AWDChannelGroup:</b> This parameter can be one of the following values: (1) On STM32L0, parameter not available on all devices: only on STM32L053xx, STM32L063xx, STM32L073xx, STM32L083xx.           <ul style="list-style-type: none"> <li>- LL_ADC_AWD_DISABLE</li> <li>- LL_ADC_AWD_ALL_CHANNELS_REG</li> <li>- LL_ADC_AWD_CHANNEL_0_REG</li> <li>- LL_ADC_AWD_CHANNEL_1_REG</li> <li>- LL_ADC_AWD_CHANNEL_2_REG</li> <li>- LL_ADC_AWD_CHANNEL_3_REG</li> <li>- LL_ADC_AWD_CHANNEL_4_REG</li> <li>- LL_ADC_AWD_CHANNEL_5_REG</li> <li>- LL_ADC_AWD_CHANNEL_6_REG</li> <li>- LL_ADC_AWD_CHANNEL_7_REG</li> <li>- LL_ADC_AWD_CHANNEL_8_REG</li> <li>- LL_ADC_AWD_CHANNEL_9_REG</li> <li>- LL_ADC_AWD_CHANNEL_10_REG</li> <li>- LL_ADC_AWD_CHANNEL_11_REG</li> <li>- LL_ADC_AWD_CHANNEL_12_REG</li> <li>- LL_ADC_AWD_CHANNEL_13_REG</li> <li>- LL_ADC_AWD_CHANNEL_14_REG</li> <li>- LL_ADC_AWD_CHANNEL_15_REG</li> <li>- LL_ADC_AWD_CHANNEL_16_REG (1)</li> <li>- LL_ADC_AWD_CHANNEL_17_REG</li> <li>- LL_ADC_AWD_CHANNEL_18_REG</li> <li>- LL_ADC_AWD_CH_VREFINT_REG</li> <li>- LL_ADC_AWD_CH_TEMPSENSOR_REG</li> <li>- LL_ADC_AWD_CH_VLCD_REG (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once monitored channels are selected, analog watchdog is enabled.</li> <li>• In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro <code>__LL_ADC_ANALOGWD_CHANNEL_GROUP()</code>.</li> </ul>

- 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 group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to  
LL API cross  
reference:

- CFGR1 AWDCH LL\_ADC\_SetAnalogWDMonitChannels
- CFGR1 AWDSGL LL\_ADC\_SetAnalogWDMonitChannels
- CFGR1 AWDEN LL\_ADC\_SetAnalogWDMonitChannels

### **LL\_ADC\_GetAnalogWDMonitChannels**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC analog watchdog monitored channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_AWD_DISABLE</li> <li>- LL_ADC_AWD_ALL_CHANNELS_REG</li> <li>- LL_ADC_AWD_CHANNEL_0_REG</li> <li>- LL_ADC_AWD_CHANNEL_1_REG</li> <li>- LL_ADC_AWD_CHANNEL_2_REG</li> <li>- LL_ADC_AWD_CHANNEL_3_REG</li> <li>- LL_ADC_AWD_CHANNEL_4_REG</li> <li>- LL_ADC_AWD_CHANNEL_5_REG</li> <li>- LL_ADC_AWD_CHANNEL_6_REG</li> <li>- LL_ADC_AWD_CHANNEL_7_REG</li> <li>- LL_ADC_AWD_CHANNEL_8_REG</li> <li>- LL_ADC_AWD_CHANNEL_9_REG</li> <li>- LL_ADC_AWD_CHANNEL_10_REG</li> <li>- LL_ADC_AWD_CHANNEL_11_REG</li> <li>- LL_ADC_AWD_CHANNEL_12_REG</li> <li>- LL_ADC_AWD_CHANNEL_13_REG</li> <li>- LL_ADC_AWD_CHANNEL_14_REG</li> <li>- LL_ADC_AWD_CHANNEL_15_REG</li> <li>- LL_ADC_AWD_CHANNEL_16_REG</li> <li>- LL_ADC_AWD_CHANNEL_17_REG</li> <li>- LL_ADC_AWD_CHANNEL_18_REG</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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 <code>__LL_ADC_CHANNEL_TO_DECIMAL_NB()</code>. 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 <code>__LL_ADC_CHANNEL_TO_DECIMAL_NB()</code>.</li> </ul>

	<p>Applicable only when the analog watchdog is set to monitor one channel.</p> <ul style="list-style-type: none"> <li>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 group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).</li> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR1 AWDCH LL_ADC_GetAnalogWDMonitChannels</li> <li>CFGTR1 AWDSGL LL_ADC_GetAnalogWDMonitChannels</li> <li>CFGTR1 AWDEN LL_ADC_GetAnalogWDMonitChannels</li> </ul>

### LL\_ADC\_ConfigAnalogWDThresholds

Function Name	<code>__STATIC_INLINE void LL_ADC_ConfigAnalogWDThresholds(ADC_TypeDef * ADCx, uint32_t AWDThresholdHighValue, uint32_t AWDThresholdLowValue)</code>
Function Description	Set ADC analog watchdog thresholds value of both thresholds high and low.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> <li><b>AWDThresholdHighValue:</b> Value between Min_Data=0x000 and Max_Data=0xFFFF</li> <li><b>AWDThresholdLowValue:</b> Value between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>If value of only one threshold high or low must be set, use function <code>LL_ADC_SetAnalogWDThresholds()</code>.</li> <li>In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro <code>__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION()</code>.</li> <li>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 group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).</li> <li>On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TR HT LL_ADC_ConfigAnalogWDThresholds</li> <li>TR LT LL_ADC_ConfigAnalogWDThresholds</li> </ul>

### LL\_ADC\_SetAnalogWDThresholds

Function Name	<code>__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds(ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)</code>
---------------	---

Function Description	Set ADC analog watchdog threshold value of threshold high or low.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>AWDThresholdsHighLow:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_AWD_THRESHOLD_HIGH</li> <li>– LL_ADC_AWD_THRESHOLD_LOW</li> </ul> </li> <li>• <b>AWDThresholdValue:</b> Value between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Return values	• <b>None:</b>
Notes	<ul style="list-style-type: none"> <li>• If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds().</li> <li>• In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().</li> <li>• 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 group regular.resolution: resolution is not limited (corresponds to ADC resolution configured).</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TR HT LL_ADC_SetAnalogWDThresholds</li> <li>• TR LT LL_ADC_SetAnalogWDThresholds</li> </ul>

### LL\_ADC\_GetAnalogWDThresholds

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds(ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow)</code>
Function Description	Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>AWDThresholdsHighLow:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_ADC_AWD_THRESHOLD_HIGH</li> <li>– LL_ADC_AWD_THRESHOLD_LOW</li> <li>– LL_ADC_AWD_THRESHOLDS_HIGH_LOW</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro: <code>__LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW()</code>.</li> <li>• In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro</li> </ul>

---

`_LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION()`.

Reference Manual to  
LL API cross  
reference:

- TR HT LL\_ADC\_GetAnalogWDThresholds
- TR LT LL\_ADC\_GetAnalogWDThresholds

### LL\_ADC\_SetOverSamplingScope

Function Name	<code>_STATIC_INLINE void LL_ADC_SetOverSamplingScope(ADC_TypeDef * ADCx, uint32_t OvsScope)</code>
Function Description	Set ADC oversampling scope.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>OvsScope:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_ADC_OVS_DISABLE</code></li> <li>- <code>LL_ADC_OVS_GRP_REGULAR_CONTINUED</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR2 OVSE LL_ADC_SetOverSamplingScope</li> </ul>

### LL\_ADC\_GetOverSamplingScope

Function Name	<code>_STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC oversampling scope.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_ADC_OVS_DISABLE</code></li> <li>- <code>LL_ADC_OVS_GRP_REGULAR_CONTINUED</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR2 OVSE LL_ADC_GetOverSamplingScope</li> </ul>

### LL\_ADC\_SetOverSamplingDiscont

Function Name	<code>_STATIC_INLINE void LL_ADC_SetOverSamplingDiscont(ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)</code>
Function Description	Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>OverSamplingDiscont:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_ADC_OVS_REG_CONT</code></li> </ul> </li> </ul>

	– LL_ADC_OVS_REG_DISCONT
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR2 TOVS LL_ADC_SetOverSamplingDiscont</li> </ul>

### LL\_ADC\_GetOverSamplingDiscont

Function Name	<b>_STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (ADC_TypeDef * ADCx)</b>
Function Description	Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_ADC_OVS_REG_CONT</li> <li>– LL_ADC_OVS_REG_DISCONT</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR2 TOVS LL_ADC_GetOverSamplingDiscont</li> </ul>

### LL\_ADC\_ConfigOverSamplingRatioShift

Function Name	<b>_STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)</b>
Function Description	Set ADC oversampling.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>Ratio:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_ADC_OVS_RATIO_2</li> <li>– LL_ADC_OVS_RATIO_4</li> <li>– LL_ADC_OVS_RATIO_8</li> <li>– LL_ADC_OVS_RATIO_16</li> <li>– LL_ADC_OVS_RATIO_32</li> <li>– LL_ADC_OVS_RATIO_64</li> <li>– LL_ADC_OVS_RATIO_128</li> <li>– LL_ADC_OVS_RATIO_256</li> </ul> </li> <li>• <b>Shift:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_ADC_OVS_SHIFT_NONE</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_ADC_OVS_SHIFT_RIGHT_1</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_2</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_3</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_4</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_5</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_6</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_7</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_8</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function set the 2 items of oversampling configuration: ratioshift</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift</li> <li>• CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift</li> </ul>

### LL\_ADC\_GetOverSamplingRatio

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC oversampling ratio.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Ratio:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_OVS_RATIO_2</li> <li>- LL_ADC_OVS_RATIO_4</li> <li>- LL_ADC_OVS_RATIO_8</li> <li>- LL_ADC_OVS_RATIO_16</li> <li>- LL_ADC_OVS_RATIO_32</li> <li>- LL_ADC_OVS_RATIO_64</li> <li>- LL_ADC_OVS_RATIO_128</li> <li>- LL_ADC_OVS_RATIO_256</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR2 OVSR LL_ADC_GetOverSamplingRatio</li> </ul>

### LL\_ADC\_GetOverSamplingShift

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift(ADC_TypeDef * ADCx)</code>
Function Description	Get ADC oversampling shift.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Shift:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_ADC_OVS_SHIFT_NONE</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_1</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_2</li> <li>- LL_ADC_OVS_SHIFT_RIGHT_3</li> </ul> </li> </ul>

- LL\_ADC\_OVS\_SHIFT\_RIGHT\_4
- LL\_ADC\_OVS\_SHIFT\_RIGHT\_5
- LL\_ADC\_OVS\_SHIFT\_RIGHT\_6
- LL\_ADC\_OVS\_SHIFT\_RIGHT\_7
- LL\_ADC\_OVS\_SHIFT\_RIGHT\_8

Reference Manual to  
LL API cross  
reference:

- CFGR2 OVSS LL\_ADC\_GetOverSamplingShift

### **LL\_ADC\_EnableInternalRegulator**

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableInternalRegulator(ADC_TypeDef * ADCx)</code></b>
Function Description	Enable ADC instance internal voltage regulator.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, there are three possibilities to enable the voltage regulator: by enabling it manually using this function (LL_ADC_EnableInternalRegulator() ).by launching a calibration using function LL_ADC_StartCalibration().by enabling the ADC using function LL_ADC_Enable().</li> <li>• On this STM32 serie, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter tUP_LDO. Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US.</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADVREGEN LL_ADC_EnableInternalRegulator</li> </ul>

### **LL\_ADC\_DisableInternalRegulator**

Function Name	<b><code>_STATIC_INLINE void LL_ADC_DisableInternalRegulator(ADC_TypeDef * ADCx)</code></b>
Function Description	Disable ADC internal voltage regulator.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADVREGEN LL_ADC_DisableInternalRegulator</li> </ul>

**LL\_ADC\_IsInternalRegulatorEnabled**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (ADC_TypeDef * ADCx)</code>
Function Description	Get the selected ADC instance internal voltage regulator state.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>0:</b> internal regulator is disabled, 1: internal regulator is enabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled</li> </ul>

**LL\_ADC\_Enable**

Function Name	<code>__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)</code>
Function Description	Enable the selected ADC instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)</li> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADEN LL_ADC_Enable</li> </ul>

**LL\_ADC\_Disable**

Function Name	<code>__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)</code>
Function Description	Disable the selected ADC instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on group regular.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADDIS LL_ADC_Disable</li> </ul>

**LL\_ADC\_IsEnabled**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)</code>
Function Description	Get the selected ADC instance enable state.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>0:</b> ADC is disabled, 1: ADC is enabled.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADEN LL_ADC_IsEnabled</li> </ul>

**LL\_ADC\_IsDisableOngoing**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (ADC_TypeDef * ADCx)</code>
Function Description	Get the selected ADC instance disable state.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>0:</b> no ADC disable command on going.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ADDIS LL_ADC_IsDisableOngoing</li> </ul>

**LL\_ADC\_StartCalibration**

Function Name	<code>__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx)</code>
Function Description	Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal <code>LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES</code>.</li> <li>• In case of usage of ADC with DMA transfer: On this STM32 serie, ADC DMA transfer request should be disabled during calibration: Calibration factor is available in data register and also transferred by DMA. To not insert ADC calibration factor among ADC conversion data in array variable, DMA transfer must be disabled during calibration. (DMA transfer setting backup and disable before calibration, DMA transfer setting restore after calibration. Refer to functions <code>LL_ADC_REG_GetDMATransfer()</code>, <code>LL_ADC_REG_SetDMATransfer()</code> ).</li> </ul>

Reference Manual to  
LL API cross  
reference:

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

- CR ADCAL 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:  
• CR ADCAL LL\_ADC\_IsCalibrationOnGoing

### **LL\_ADC\_REG\_StartConversion**

Function Name      **`_STATIC_INLINE void LL_ADC_REG_StartConversion(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 for both internal trigger (SW start) and 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 will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.  
 • On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular.

Reference Manual to  
LL API cross  
reference:  
• CR ADSTART LL\_ADC\_REG\_StartConversion

### **LL\_ADC\_REG\_StopConversion**

Function Name      **`_STATIC_INLINE void LL_ADC_REG_StopConversion(ADC_TypeDef * ADCx)`**

Function Description      Stop ADC group regular conversion.

Parameters      • **ADCx:** ADC instance

Return values      • **None:**

Notes      • On this STM32 serie, setting of this feature is conditioned to

ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

Reference Manual to  
LL API cross  
reference:

- CR ADSTP LL\_ADC\_REG\_StopConversion

### **LL\_ADC\_REG\_IsConversionOngoing**

Function Name      **`__STATIC_INLINE uint32_t  
LL_ADC_REG_IsConversionOngoing (ADC_TypeDef * ADCx)`**

Function Description      Get ADC group regular conversion state.

Parameters      • **ADCx:** ADC instance

Return values      • **0:** no conversion is on going on ADC group regular.

Reference Manual to  
LL API cross  
reference:

### **LL\_ADC\_REG\_IsStopConversionOngoing**

Function Name      **`__STATIC_INLINE uint32_t  
LL_ADC_REG_IsStopConversionOngoing (ADC_TypeDef *  
ADCx)`**

Function Description      Get ADC group regular command of conversion stop state.

Parameters      • **ADCx:** ADC instance

Return values      • **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to  
LL API cross  
reference:

### **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:

**LL\_ADC\_REG\_ReadConversionData12**

Function Name	<code>__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion data, range fit for ADC resolution 12 bits.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DATA LL_ADC_REG_ReadConversionData12</li> </ul>

**LL\_ADC\_REG\_ReadConversionData10**

Function Name	<code>__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion data, range fit for ADC resolution 10 bits.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x000 and Max_Data=0x3FF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DATA LL_ADC_REG_ReadConversionData10</li> </ul>

**LL\_ADC\_REG\_ReadConversionData8**

Function Name	<code>__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion data, range fit for ADC resolution 8 bits.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DATA LL_ADC_REG_ReadConversionData8</li> </ul>

**LL\_ADC\_REG\_ReadConversionData6**

Function Name	<code>__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (ADC_TypeDef * ADCx)</code>
Function Description	Get ADC group regular conversion data, range fit for ADC resolution 6 bits.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x3F</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DATA LL_ADC_REG_ReadConversionData6</li> </ul>

**LL\_ADC\_IsActiveFlag\_ADRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (ADC_TypeDef * ADCx)</code>
Function Description	Get flag ADC ready.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ADRDY LL_ADC_IsActiveFlag_ADRDY</li> </ul>

**LL\_ADC\_IsActiveFlag\_EOC**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (ADC_TypeDef * ADCx)</code>
Function Description	Get flag ADC group regular end of unitary conversion.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

Reference Manual to LL API cross reference:

**LL\_ADC\_IsActiveFlag\_EOS**

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (ADC_TypeDef * ADCx)</code>
Function Description	Get flag ADC group regular end of sequence conversions.

---

Parameters	<ul style="list-style-type: none"><li><b>ADCx:</b> ADC instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• ISR EOSEQ LL_ADC_IsActiveFlag_EOS

### LL\_ADC\_IsActiveFlag\_OVR

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR(ADC_TypeDef * ADCx)</code></b>
Function Description	Get flag ADC group regular overrun.
Parameters	<ul style="list-style-type: none"><li><b>ADCx:</b> ADC instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• ISR OVR LL_ADC_IsActiveFlag_OVR

### LL\_ADC\_IsActiveFlag\_EOSMP

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP(ADC_TypeDef * ADCx)</code></b>
Function Description	Get flag ADC group regular end of sampling phase.
Parameters	<ul style="list-style-type: none"><li><b>ADCx:</b> ADC instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

### LL\_ADC\_IsActiveFlag\_AWD1

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1(ADC_TypeDef * ADCx)</code></b>
Function Description	Get flag ADC analog watchdog 1 flag.
Parameters	<ul style="list-style-type: none"><li><b>ADCx:</b> ADC instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• ISR AWD LL_ADC_IsActiveFlag_AWD1

### LL\_ADC\_IsActiveFlag\_EOCAL

Function Name	<b><code>_STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOCAL(ADC_TypeDef * ADCx)</code></b>
Function Description	Get flag ADC end of calibration.
Parameters	<ul style="list-style-type: none"><li><b>ADCx:</b> ADC instance</li></ul>

---

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR EOCAL LL_ADC_IsActiveFlag_EOCAL</li> </ul>

### LL\_ADC\_ClearFlag\_ADRDY

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY(ADC_TypeDef * ADCx)</code>
Function Description	Clear flag ADC ready.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR ADRDY LL_ADC_ClearFlag_ADRDY</li> </ul>

### LL\_ADC\_ClearFlag\_EOC

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_EOC(ADC_TypeDef * ADCx)</code>
Function Description	Clear flag ADC group regular end of unitary conversion.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR EOC LL_ADC_ClearFlag_EOC</li> </ul>

### LL\_ADC\_ClearFlag\_EOS

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_EOS(ADC_TypeDef * ADCx)</code>
Function Description	Clear flag ADC group regular end of sequence conversions.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR EOSEQ LL_ADC_ClearFlag_EOS</li> </ul>

### LL\_ADC\_ClearFlag\_OVR

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_OVR(ADC_TypeDef * ADCx)</code>
---------------	--

---

Function Description	Clear flag ADC group regular overrun.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR OVR LL_ADC_ClearFlag_OVR</li> </ul>

### LL\_ADC\_ClearFlag\_EOSMP

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP(ADC_TypeDef * ADCx)</code>
Function Description	Clear flag ADC group regular end of sampling phase.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR EOSMP LL_ADC_ClearFlag_EOSMP</li> </ul>

### LL\_ADC\_ClearFlag\_AWD1

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_AWD1(ADC_TypeDef * ADCx)</code>
Function Description	Clear flag ADC analog watchdog 1.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR AWD LL_ADC_ClearFlag_AWD1</li> </ul>

### LL\_ADC\_ClearFlag\_EOCAL

Function Name	<code>__STATIC_INLINE void LL_ADC_ClearFlag_EOCAL(ADC_TypeDef * ADCx)</code>
Function Description	Clear flag ADC end of calibration.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR EOCAL LL_ADC_ClearFlag_EOCAL</li> </ul>

### LL\_ADC\_EnableIT\_ADRDY

Function Name	<code>__STATIC_INLINE void LL_ADC_EnableIT_ADRDY(ADC_TypeDef * ADCx)</code>
Function Description	Enable ADC ready.

---

Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER ADRDYIE LL_ADC_EnableIT_ADRDY</li> </ul>

### LL\_ADC\_EnableIT\_EOC

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)</code></b>
Function Description	Enable interruption ADC group regular end of unitary conversion.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOCIE LL_ADC_EnableIT_EOC</li> </ul>

### LL\_ADC\_EnableIT\_EOS

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)</code></b>
Function Description	Enable interruption ADC group regular end of sequence conversions.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOSEQIE LL_ADC_EnableIT_EOS</li> </ul>

### LL\_ADC\_EnableIT\_OVR

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)</code></b>
Function Description	Enable ADC group regular interruption overrun.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER OVRIE LL_ADC_EnableIT_OVR</li> </ul>

### LL\_ADC\_EnableIT\_EOSMP

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)</code></b>
Function Description	Enable interruption ADC group regular end of sampling.

---

Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOSMPIE LL_ADC_EnableIT_EOSMP</li> </ul>

### LL\_ADC\_EnableIT\_AWD1

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableIT_AWD1(ADC_TypeDef * ADCx)</code></b>
Function Description	Enable interruption ADC analog watchdog 1.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER AWDIE LL_ADC_EnableIT_AWD1</li> </ul>

### LL\_ADC\_EnableIT\_EOCAL

Function Name	<b><code>_STATIC_INLINE void LL_ADC_EnableIT_EOCAL(ADC_TypeDef * ADCx)</code></b>
Function Description	Enable interruption ADC end of calibration.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOCALEIE LL_ADC_EnableIT_EOCAL</li> </ul>

### LL\_ADC\_DisableIT\_ADRDY

Function Name	<b><code>_STATIC_INLINE void LL_ADC_DisableIT_ADRDY(ADC_TypeDef * ADCx)</code></b>
Function Description	Disable interruption ADC ready.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER ADRDYIE LL_ADC_DisableIT_ADRDY</li> </ul>

### LL\_ADC\_DisableIT\_EOC

Function Name	<b><code>_STATIC_INLINE void LL_ADC_DisableIT_EOC(ADC_TypeDef * ADCx)</code></b>
Function Description	Disable interruption ADC group regular end of unitary conversion.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOCIE LL_ADC_DisableIT_EOC</li> </ul>

### LL\_ADC\_DisableIT\_EOS

Function Name	<b><u>__STATIC_INLINE void LL_ADC_DisableIT_EOS(ADC_TypeDef * ADCx)</u></b>
Function Description	Disable interruption ADC group regular end of sequence conversions.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOSEQIE LL_ADC_DisableIT_EOS</li> </ul>

### LL\_ADC\_DisableIT\_OVR

Function Name	<b><u>__STATIC_INLINE void LL_ADC_DisableIT_OVR(ADC_TypeDef * ADCx)</u></b>
Function Description	Disable interruption ADC group regular overrun.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER OVRIE LL_ADC_DisableIT_OVR</li> </ul>

### LL\_ADC\_DisableIT\_EOSMP

Function Name	<b><u>__STATIC_INLINE void LL_ADC_DisableIT_EOSMP(ADC_TypeDef * ADCx)</u></b>
Function Description	Disable interruption ADC group regular end of sampling.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOSMPIE LL_ADC_DisableIT_EOSMP</li> </ul>

### LL\_ADC\_DisableIT\_AWD1

Function Name	<b><u>__STATIC_INLINE void LL_ADC_DisableIT_AWD1(ADC_TypeDef * ADCx)</u></b>
Function Description	Disable interruption ADC analog watchdog 1.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER AWDIE LL_ADC_DisableIT_AWD1</li> </ul>

### LL\_ADC\_DisableIT\_EOCAL

Function Name	<code>__STATIC_INLINE void LL_ADC_DisableIT_EOCAL(ADC_TypeDef * ADCx)</code>
Function Description	Disable interruption ADC end of calibration.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOCALIE LL_ADC_DisableIT_EOCAL</li> </ul>

### LL\_ADC\_IsEnabledIT\_ADRDY

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY(ADC_TypeDef * ADCx)</code>
Function Description	Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY</li> </ul>

### LL\_ADC\_IsEnabledIT\_EOC

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC(ADC_TypeDef * ADCx)</code>
Function Description	Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOCIE LL_ADC_IsEnabledIT_EOC</li> </ul>

### LL\_ADC\_IsEnabledIT\_EOS

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS(ADC_TypeDef * ADCx)</code>
Function Description	Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOSEQIE LL_ADC_IsEnabledIT_EOS</li> </ul>

### LL\_ADC\_IsEnabledIT\_OVR

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR(ADC_TypeDef * ADCx)</code>
Function Description	Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER OVRIE LL_ADC_IsEnabledIT_OVR</li> </ul>

### LL\_ADC\_IsEnabledIT\_EOSMP

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP(ADC_TypeDef * ADCx)</code>
Function Description	Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP</li> </ul>

### LL\_ADC\_IsEnabledIT\_AWD1

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1(ADC_TypeDef * ADCx)</code>
Function Description	Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IER AWDIE LL_ADC_IsEnabledIT_AWD1</li> </ul>

### LL\_ADC\_IsEnabledIT\_EOCAL

Function Name	<code>__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOCAL(ADC_TypeDef * ADCx)</code>
---------------	--

**Function Description** Get state of interruption ADC end of calibration (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:**

- IER EOCALIE LL\_ADC\_IsEnabledIT\_EOCAL

### 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

**Notes**

- This function is performing a hard reset, using high level clock source RCC ADC reset.

### 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	<ul style="list-style-type: none"> <li><b>ADC_CommonInitStruct:</b> Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
<b>LL_ADC_Delnit</b>	
Function Name	<b>ErrorStatus LL_ADC_Delnit (ADC_TypeDef * ADCx)</b>
Function Description	De-initialize registers of the selected ADC instance to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:             <ul style="list-style-type: none"> <li>SUCCESS: ADC registers are de-initialized</li> <li>ERROR: ADC registers are not de-initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDelnit().</li> <li>If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Refer to function LL_ADC_CommonDelnit().</li> </ul>
<b>LL_ADC_Init</b>	
Function Name	<b>ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)</b>
Function Description	Initialize some features of ADC instance.
Parameters	<ul style="list-style-type: none"> <li><b>ADCx:</b> ADC instance</li> <li><b>ADC_InitStruct:</b> Pointer to a LL_ADC_REG_InitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:             <ul style="list-style-type: none"> <li>SUCCESS: ADC registers are initialized</li> <li>ERROR: ADC registers are not initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>These parameters have an impact on ADC scope: ADC instance. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .</li> <li>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.</li> <li>After using this function, some other features must be configured using LL unitary functions. The minimum</li> </ul>

configuration remaining to be done is: Set ADC group regular sequencer: map channel on rank corresponding to channel number. Refer to function  
**LL\_ADC\_REG\_SetSequencerChannels();** Set ADC channel sampling time Refer to function  
**LL\_ADC\_SetChannelSamplingTime();**

### **LL\_ADC\_StructInit**

Function Name	<b>void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)</b>
Function Description	Set each <b>LL_ADC_InitTypeDef</b> field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADC_InitStruct:</b> Pointer to a <b>LL_ADC_InitTypeDef</b> structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### **LL\_ADC\_REG\_Init**

Function Name	<b>ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_REG_InitStruct)</b>
Function Description	Initialize some features of ADC group regular.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADCx:</b> ADC instance</li> <li>• <b>ADC_REG_InitStruct:</b> Pointer to a <b>LL_ADC_REG_InitTypeDef</b> structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: ADC registers are initialized</li> <li>– ERROR: ADC registers are not initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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").</li> <li>• The setting of these parameters by function <b>LL_ADC_Init()</b> 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.</li> <li>• 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 sequencer: map channel on rank corresponding to channel number. Refer to function  <b>LL_ADC_REG_SetSequencerChannels();</b> Set ADC channel sampling time Refer to function  <b>LL_ADC_SetChannelSamplingTime();</b></li> </ul>

**LL\_ADC\_REG\_StructInit**

Function Name	<code>void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)</code>
Function Description	Set each <code>LL_ADC_REG_InitTypeDef</code> field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>ADC_REG_InitStruct:</b> Pointer to a <code>LL_ADC_REG_InitTypeDef</code> structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 53.3 ADC Firmware driver defines

### 53.3.1 ADC

#### *Analog watchdog - Monitored channels*

<code>LL_ADC_AWD_DISABLE</code>	ADC analog watchdog monitoring disabled
<code>LL_ADC_AWD_ALL_CHANNELS_REG</code>	ADC analog watchdog monitoring of all channels, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_0_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_1_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_2_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_3_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_4_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_5_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_6_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group regular only
<code>LL_ADC_AWD_CHANNEL_7_REG</code>	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO

LL_ADC_AWD_CHANNEL_8_REG	pin) ADCx_IN7, converted by group regular only
LL_ADC_AWD_CHANNEL_9_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_10_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_11_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_12_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_13_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_14_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_15_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_17_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_18_REG	ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only
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_TEMPSENSOR_REG	ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only

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_CH_VLCD_REG	ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda, converted by group regular only

**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
LL_ADC_AWD_THRESHOLDS_HIGH_LOW	ADC analog watchdog both thresholds high and low concatenated into the same data

**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_17	ADC external channel (channel connected to GPIO pin) ADCx_IN17
LL_ADC_CHANNEL_18	ADC external channel (channel connected to GPIO pin) ADCx_IN18
LL_ADC_CHANNEL_VREFINT	ADC internal channel connected to VrefInt: Internal voltage reference.
LL_ADC_CHANNEL_TEMPSENSOR	ADC internal channel connected to Temperature sensor.
LL_ADC_CHANNEL_16	ADC external channel (channel connected to GPIO pin) ADCx_IN16
LL_ADC_CHANNEL_VLCD	ADC internal channel connected to Vlcd: Vlcd voltage through a divider ladder of factor 1/4, 1/3 or 1/2 (set by LCD voltage generator biasing), to have Vlcd always below Vdda.

***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 instance - Clock source***

LL_ADC_CLOCK_SYNC_PCLK_DIV4	ADC synchronous clock derived from AHB clock divided by 4
LL_ADC_CLOCK_SYNC_PCLK_DIV2	ADC synchronous clock derived from AHB clock divided by 2
LL_ADC_CLOCK_SYNC_PCLK_DIV1	ADC synchronous clock derived from AHB clock not divided
LL_ADC_CLOCK_ASYNC	ADC asynchronous clock. Asynchronous clock prescaler can be configured using function

***ADC common - Clock frequency mode***

LL_ADC_CLOCK_FREQ_MODE_HIGH	ADC clock mode to high frequency. On STM32L0, ADC clock frequency above 2.8MHz.
LL_ADC_CLOCK_FREQ_MODE_LOW	ADC clock mode to low frequency. On STM32L0, ADC clock frequency below 2.8MHz.

**ADC common - Clock source**

LL_ADC_CLOCK_ASYNC_DIV1	ADC asynchronous clock without prescaler
LL_ADC_CLOCK_ASYNC_DIV2	ADC asynchronous clock with prescaler division by 2. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV4	ADC asynchronous clock with prescaler division by 4. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV6	ADC asynchronous clock with prescaler division by 6. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV8	ADC asynchronous clock with prescaler division by 8. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV10	ADC asynchronous clock with prescaler division by 10. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV12	ADC asynchronous clock with prescaler division by 12. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV16	ADC asynchronous clock with prescaler division by 16. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV32	ADC asynchronous clock with prescaler division by 32. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV64	ADC asynchronous clock with prescaler division by 64. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV128	ADC asynchronous clock with prescaler division by 128. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding

	ADC instance clock is set to clock source asynchronous (refer to function)
LL_ADC_CLOCK_ASYNC_DIV256	ADC asynchronous clock with prescaler division by 256. ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous (refer to function)

***ADC common - Measurement path to internal channels***

LL_ADC_PATH_INTERNAL_NONE	ADC measurement paths 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
LL_ADC_PATH_INTERNAL_VLCD	ADC measurement path to internal channel Vlcd

***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 (alignment on data register MSB bit 15)

***ADC flags***

LL_ADC_FLAG_ADRDY	ADC flag ADC instance ready
LL_ADC_FLAG_EOC	ADC flag ADC group regular end of unitary conversion
LL_ADC_FLAG_EOS	ADC flag ADC group regular end of sequence conversions
LL_ADC_FLAG_OVR	ADC flag ADC group regular overrun
LL_ADC_FLAG_EOSMP	ADC flag ADC group regular end of sampling phase
LL_ADC_FLAG_AWD1	ADC flag ADC analog watchdog 1
LL_ADC_FLAG_EOCAL	ADC flag end of calibration

***ADC instance - Groups***

LL_ADC_GROUP_REGULAR	ADC group regular (available on all STM32 devices)
----------------------	--

***Definitions of ADC hardware constraints delays***

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US	Delay for ADC stabilization time (ADC voltage regulator start-up time)
LL_ADC_DELAY_VREFINT_STAB_US	Delay for internal voltage reference stabilization time
LL_ADC_DELAY_TEMPSENSOR_STAB_US	Delay for temperature sensor stabilization time
LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES	Delay required between ADC end of calibration and ADC enable

***ADC interruptions for configuration (interrupt enable or disable)***

LL_ADC_IT_ADRDY	ADC interruption ADC instance ready
-----------------	-------------------------------------

LL_ADC_IT_EOC	ADC interruption ADC group regular end of unitary conversion
LL_ADC_IT_EOS	ADC interruption ADC group regular end of sequence conversions
LL_ADC_IT_OVR	ADC interruption ADC group regular overrun
LL_ADC_IT_EOSMP	ADC interruption ADC group regular end of sampling phase
LL_ADC_IT_AWD1	ADC interruption ADC analog watchdog 1
LL_ADC_IT_EOCAL	ADC interruption ADC end of calibration
<b><i>ADC instance - Low power mode</i></b>	
LL_ADC_LP_MODE_NONE	No ADC low power mode activated
LL_ADC_LP_AUTOWAIT	ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function
LL_ADC_LP_AUTOPOWEROFF	ADC low power mode auto power-off: the ADC automatically powers-off after a ADC conversion and automatically wakes up when a new ADC conversion is triggered (with startup time between trigger and start of sampling). See description with function
LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF	ADC low power modes auto wait and auto power-off combined. See description with function
<b><i>Oversampling - Discontinuous mode</i></b>	
LL_ADC_OVS_REG_CONT	ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)
LL_ADC_OVS_REG_DISCONT	ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)
<b><i>Oversampling - Ratio</i></b>	
LL_ADC_OVS_RATIO_2	ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_4	ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_8	ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_16	ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to

	result as the ADC oversampling conversion data (before potential shift)
LL_ADC_OVS_RATIO_32	ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift))
LL_ADC_OVS_RATIO_64	ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift))
LL_ADC_OVS_RATIO_128	ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift))
LL_ADC_OVS_RATIO_256	ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift))

#### ***Oversampling - Oversampling scope***

LL_ADC_OVS_DISABLE	ADC oversampling disabled.
LL_ADC_OVS_GRP_REGULAR_CONTINUED	ADC oversampling on conversions of ADC group regular. Literal suffix "continued" is kept for compatibility with other STM32 devices featuring ADC group injected, in this case other oversampling scope parameters are available.

#### ***Oversampling - Data shift***

LL_ADC_OVS_SHIFT_NONE	ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_1	ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_2	ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_3	ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_4	ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_5	ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_6	ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC

	oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_7	ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)
LL_ADC_OVS_SHIFT_RIGHT_8	ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

***ADC registers compliant with specific purpose***

LL\_ADC\_DMA\_REG\_REGULAR\_DATA

***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_LIMITED	ADC conversion data are transferred by DMA, in 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.
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 - Overrun behavior on conversion data***

LL_ADC_REG_OVR_DATA_PRESERVED	ADC group regular behavior in case of overrun: data preserved
LL_ADC_REG_OVR_DATA_OVERWRITTEN	ADC group regular behavior in case of overrun: data overwritten

***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

***ADC group regular - Sequencer scan direction***

LL_ADC_REG_SEQ_SCAN_DIR_FORWARD	ADC group regular sequencer scan direction forward: from lowest channel
---------------------------------	---

number to highest channel number (scan of all ranks, ADC conversion of ranks with channels enabled in sequencer). On some other STM32 families, this setting is not available and the default scan direction is forward.

LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD	ADC group regular sequencer scan direction backward: from highest channel number to lowest channel number (scan of all ranks, ADC conversion of ranks with channels enabled in sequencer)
----------------------------------	---

***ADC group regular - Trigger edge***

LL_ADC_REG_TRIG_EXT_RISING	ADC group regular conversion trigger polarity set to rising edge
LL_ADC_REG_TRIG_EXT_FALLING	ADC group regular conversion trigger polarity set to falling edge
LL_ADC_REG_TRIG_EXT_RISINGFALLING	ADC group regular conversion trigger polarity set to both rising and falling edges

***ADC group regular - Trigger source***

LL_ADC_REG_TRIG_SOFTWARE	ADC group regular conversion trigger internal: SW start.
LL_ADC_REG_TRIG_EXT_TIM6_TRGO	ADC group regular conversion trigger from external IP: TIM6 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM21_CH2	ADC group regular conversion trigger from external IP: TIM21 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM2_TRGO	ADC group regular conversion trigger from external IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM2_CH4	ADC group regular conversion trigger from external IP: TIM2 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).
LL_ADC_REG_TRIG_EXT_TIM22_TRGO	ADC group regular conversion trigger from external IP: TIM22 TRGO. 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 4 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 TRG0. Trigger edge set to rising edge (default setting).

`LL_ADC_REG_TRIG_EXT EXTI_LINE11` ADC group regular conversion trigger external interrupt line 11. Trigger edge set to rising edge (default setting).

#### ***ADC instance - Resolution***

<code>LL_ADC_RESOLUTION_12B</code>	ADC resolution 12 bits
<code>LL_ADC_RESOLUTION_10B</code>	ADC resolution 10 bits
<code>LL_ADC_RESOLUTION_8B</code>	ADC resolution 8 bits
<code>LL_ADC_RESOLUTION_6B</code>	ADC resolution 6 bits

#### ***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` (1)
  - `LL_ADC_CHANNEL_17`
  - `LL_ADC_CHANNEL_18`
  - `LL_ADC_CHANNEL_VREFINT`
  - `LL_ADC_CHANNEL_TEMPSENSOR`
  - `LL_ADC_CHANNEL_VLCD` (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 (1)
  - LL\_ADC\_CHANNEL\_17
  - LL\_ADC\_CHANNEL\_18
  - LL\_ADC\_CHANNEL\_VREFINT (2)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (2)
  - LL\_ADC\_CHANNEL\_VLCD (1)(2)

#### Notes:

- Example:  
`__LL_ADC_DECIMAL_NB_TO_CHANNEL(4)`  
 will return a data equivalent to  
`"LL_ADC_CHANNEL_4"`.

### \_\_LL\_ADC\_IS\_CHANNEL\_INTERRUPT\_NAL

#### 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 (1)
- LL\_ADC\_CHANNEL\_17
- LL\_ADC\_CHANNEL\_18
- LL\_ADC\_CHANNEL\_VREFINT
- LL\_ADC\_CHANNEL\_TEMPSENSOR
- LL\_ADC\_CHANNEL\_VLCD (1)

**Return value:**

- -: 0 if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin)
- 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` (1)
  - `LL_ADC_CHANNEL_17`
  - `LL_ADC_CHANNEL_18`
  - `LL_ADC_CHANNEL_VREFINT`
  - `LL_ADC_CHANNEL_TEMPSENSOR`
  - `LL_ADC_CHANNEL_VLCD` (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`
  - `LL_ADC_CHANNEL_18`

**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`
  - `LL_ADC_CHANNEL_TEMPSENSOR`
  - `LL_ADC_CHANNEL_VLCD (1)`

**Return value:**

- : 0 if the internal channel selected is not available on the ADC instance selected.  
- 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 (1)
  - LL\_ADC\_CHANNEL\_17
  - LL\_ADC\_CHANNEL\_18
  - LL\_ADC\_CHANNEL\_VREFINT (2)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (2)
  - LL\_ADC\_CHANNEL\_VLCD (1)(2)
- \_\_GROUP\_\_: This parameter can be one of the following values:
    - LL\_ADC\_GROUP\_REGULAR

**Return value:**

- Returned: value can be one of the following values:
  - LL\_ADC\_AWD\_DISABLE
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_REG
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG
  - LL\_ADC\_AWD\_CHANNEL\_15\_REG
  - LL\_ADC\_AWD\_CHANNEL\_16\_REG (1)
  - LL\_ADC\_AWD\_CHANNEL\_17\_REG
  - LL\_ADC\_AWD\_CHANNEL\_18\_REG
  - LL\_ADC\_AWD\_CH\_VREFINT\_REG
  - LL\_ADC\_AWD\_CH\_TEMPSENSOR\_REG

- LL\_ADC\_AWD\_CH\_VLCD\_REG (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\_TH  
RESHOLD\_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
  - LL\_ADC\_RESOLUTION\_10B
  - LL\_ADC\_RESOLUTION\_8B
  - LL\_ADC\_RESOLUTION\_6B
- \_\_AWD\_THRESHOLD\_\_: Value between Min\_Data=0x000 and Max\_Data=0xFFFF

#### Return value:

- Value: between Min\_Data=0x000 and Max\_Data=0xFFFF

#### Notes:

- To be used with function LL\_ADC\_ConfigAnalogWDThresholds() or 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\_T  
HRESHOLD\_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

- LL\_ADC\_RESOLUTION\_10B
- LL\_ADC\_RESOLUTION\_8B
- LL\_ADC\_RESOLUTION\_6B
- AWD\_THRESHOLD\_12\_BITS: Value between Min\_Data=0x000 and Max\_Data=0xFFFF

**Return value:**

- Value: between Min\_Data=0x000 and Max\_Data=0xFFFF

**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\_ANALOGWD\_THRESHOLDS\_HIGH\_LOW**Description:**

- Helper macro to get the ADC analog watchdog threshold high or low from raw value containing both thresholds concatenated.

**Parameters:**

- AWD\_THRESHOLD\_TYPE: This parameter can be one of the following values:
  - LL\_ADC\_AWD\_THRESHOLD\_HIGH
  - LL\_ADC\_AWD\_THRESHOLD\_LOW
- AWD\_THRESHOLDS: Value between Min\_Data=0x00000000 and Max\_Data=0xFFFFFFFF

**Return value:**

- Value: between Min\_Data=0x000 and Max\_Data=0xFFFF

**Notes:**

- To be used with function LL\_ADC\_GetAnalogWDThresholds(). Example, to get analog watchdog threshold high from the register raw value: LL\_ADC\_ANALOGWD\_THRESHOLDS\_HIGH\_LOW(LL\_ADC\_AWD\_THRESHOLD\_HIGH, <raw\_value\_with\_both\_thresholds>);

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.

`_LL_ADC_IS_ENABLED_ALL_C  
OMMON_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:**

- -: 0 All ADC instances sharing the same ADC common instance are disabled.  
– 1 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).

`_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`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`

**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_CONVERT_DATA_RESOLUTION`

**Description:**

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

**Parameters:**

- `_DATA_`: ADC conversion data to be converted
- `_ADC_RESOLUTION_CURRENT_`: Resolution of the data to be converted This parameter can be one of the following values:
  - `LL_ADC_RESOLUTION_12B`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`
- `_ADC_RESOLUTION_TARGET_`: Resolution of the data after conversion This parameter can be one of the following values:
  - `LL_ADC_RESOLUTION_12B`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`

**Return value:**

- ADC: conversion data to the requested resolution

`_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`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`

**Return value:**

- ADC: conversion data equivalent voltage value

(unit: mVolt)

**Notes:**

- 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()`

`__LL_ADC_CALC_VREFANALOG_VOLTAGE`**Description:**

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

**Parameters:**

- `__VREFINT_ADC_DATA__`: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- `__ADC_RESOLUTION__`: This parameter can be one of the following values:
  - `LL_ADC_RESOLUTION_12B`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`

**Return value:**

- Analog: reference voltage (unit: mV)

**Notes:**

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 serie, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

`__LL_ADC_CALC_TEMPERATUR_E`**Description:**

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

**Parameters:**

- `__VREFANALOG_VOLTAGE__`: Analog reference 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`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`

#### Return value:

- Temperature: (unit: degree Celsius)

#### Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS\_ADC\_DATA - TS\_CAL1) \* (TS\_CAL2\_TEMP - TS\_CAL1\_TEMP)) / (TS\_CAL2 - TS\_CAL1) + TS\_CAL1\_TEMP with TS\_ADC\_DATA = temperature sensor raw data measured by ADC Avg\_Slope = (TS\_CAL2 - TS\_CAL1) / (TS\_CAL2\_TEMP - TS\_CAL1\_TEMP) TS\_CAL1 = equivalent TS\_ADC\_DATA at temperature TEMP\_DEGC\_CAL1 (calibrated in factory) TS\_CAL2 = equivalent TS\_ADC\_DATA at temperature TEMP\_DEGC\_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro

`__LL_ADC_CALC_TEMPERATURE_TYP_PAR_AMS()`. 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 either known from user board environment or can be calculated using ADC measurement and ADC helper macro

`__LL_ADC_CALC_VREFANALOG_VOLTAGE()`. On this STM32 serie, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

`__LL_ADC_CALC_TEMPERATUR_E_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 STM32L0, 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 STM32L0, refer to device datasheet parameter "V130" (corresponding to TS\_CAL2).
- `__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`
  - `LL_ADC_RESOLUTION_10B`
  - `LL_ADC_RESOLUTION_8B`
  - `LL_ADC_RESOLUTION_6B`

#### 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 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 either known from user board environment or can be calculated using ADC measurement and ADC helper macro \_\_LL\_ADC\_CALC\_VREFANALOG\_VOLTAGE(). 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

## 54 LL BUS Generic Driver

### 54.1 BUS Firmware driver API description

#### 54.1.1 Detailed description of functions

##### **LL\_AHB1\_GRP1\_EnableClock**

Function Name      **`_STATIC_INLINE void LL_AHB1_GRP1_EnableClock  
(uint32_t Periph)`**

Function Description      Enable AHB1 peripherals clock.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
    - `LL_AHB1_GRP1_PERIPH_DMA1`
    - `LL_AHB1_GRP1_PERIPH_MIF`
    - `LL_AHB1_GRP1_PERIPH_CRC`
    - `LL_AHB1_GRP1_PERIPH_TSC (*)`
    - `LL_AHB1_GRP1_PERIPH RNG (*)`
    - `LL_AHB1_GRP1_PERIPH_CRYP (*)`

Return values

- **None:**

- Reference Manual to  
LL API cross  
reference:
- `AHBENR DMAEN LL_AHB1_GRP1_EnableClock`
  - `AHBENR MIFEN LL_AHB1_GRP1_EnableClock`
  - `AHBENR CRCEN LL_AHB1_GRP1_EnableClock`
  - `AHBENR TSCEN LL_AHB1_GRP1_EnableClock`
  - `AHBENR RNGEN LL_AHB1_GRP1_EnableClock`
  - `AHBENR CRYPTEN LL_AHB1_GRP1_EnableClock`

##### **LL\_AHB1\_GRP1\_IsEnabledClock**

Function Name      **`_STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock  
(uint32_t Periph)`**

Function Description      Check if AHB1 peripheral clock is enabled or not.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
    - `LL_AHB1_GRP1_PERIPH_DMA1`
    - `LL_AHB1_GRP1_PERIPH_MIF`
    - `LL_AHB1_GRP1_PERIPH_CRC`
    - `LL_AHB1_GRP1_PERIPH_TSC (*)`
    - `LL_AHB1_GRP1_PERIPH RNG (*)`
    - `LL_AHB1_GRP1_PERIPH_CRYP (*)`

Return values

- **State:** of Periph (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- `AHBENR DMAEN LL_AHB1_GRP1_IsEnabledClock`
  - `AHBENR MIFEN LL_AHB1_GRP1_IsEnabledClock`
  - `AHBENR CRCEN LL_AHB1_GRP1_IsEnabledClock`
  - `AHBENR TSCEN LL_AHB1_GRP1_IsEnabledClock`
  - `AHBENR RNGEN LL_AHB1_GRP1_IsEnabledClock`

- AHBENR CRYPTEN LL\_AHB1\_GRP1\_IsEnabledClock

### **LL\_AHB1\_GRP1\_DisableClock**

Function Name      **\_\_STATIC\_INLINE void LL\_AHB1\_GRP1\_DisableClock (uint32\_t Periph)**

Function Description      Disable AHB1 peripherals clock.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
    - LL\_AHB1\_GRP1\_PERIPH\_DMA1
    - LL\_AHB1\_GRP1\_PERIPH\_MIF
    - LL\_AHB1\_GRP1\_PERIPH\_CRC
    - LL\_AHB1\_GRP1\_PERIPH\_TSC (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CRYP (\*)

- Return values
- **None:**

Reference Manual to  
LL API cross  
reference:

- AHBENR DMAEN LL\_AHB1\_GRP1\_DisableClock
- AHBENR MIFEN LL\_AHB1\_GRP1\_DisableClock
- AHBENR CRCEN LL\_AHB1\_GRP1\_DisableClock
- AHBENR TSCEN LL\_AHB1\_GRP1\_DisableClock
- AHBENR RNGEN LL\_AHB1\_GRP1\_DisableClock
- AHBENR CRYPTEN LL\_AHB1\_GRP1\_DisableClock

### **LL\_AHB1\_GRP1\_ForceReset**

Function Name      **\_\_STATIC\_INLINE void LL\_AHB1\_GRP1\_ForceReset (uint32\_t Periph)**

Function Description      Force AHB1 peripherals reset.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
    - LL\_AHB1\_GRP1\_PERIPH\_ALL
    - LL\_AHB1\_GRP1\_PERIPH\_DMA1
    - LL\_AHB1\_GRP1\_PERIPH\_MIF
    - LL\_AHB1\_GRP1\_PERIPH\_CRC
    - LL\_AHB1\_GRP1\_PERIPH\_TSC (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CRYP (\*)

- Return values
- **None:**

Reference Manual to  
LL API cross  
reference:

- AHBRSTR DMARST LL\_AHB1\_GRP1\_ForceReset
- AHBRSTR MIFRST LL\_AHB1\_GRP1\_ForceReset
- AHBRSTR CRCRST LL\_AHB1\_GRP1\_ForceReset
- AHBRSTR TSCRST LL\_AHB1\_GRP1\_ForceReset
- AHBRSTR Rngrst LL\_AHB1\_GRP1\_ForceReset
- AHBRSTR CRYPRST LL\_AHB1\_GRP1\_ForceReset

### **LL\_AHB1\_GRP1\_ReleaseReset**

Function Name      **\_\_STATIC\_INLINE void LL\_AHB1\_GRP1\_ReleaseReset (uint32\_t Periph)**

Function Description	Release AHB1 peripherals reset.
Parameters	<ul style="list-style-type: none"> <li><b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_AHB1_GRP1_PERIPH_ALL</li> <li>- LL_AHB1_GRP1_PERIPH_DMA1</li> <li>- LL_AHB1_GRP1_PERIPH_MIF</li> <li>- LL_AHB1_GRP1_PERIPH_CRC</li> <li>- LL_AHB1_GRP1_PERIPH_TSC (*)</li> <li>- LL_AHB1_GRP1_PERIPH_RNG (*)</li> <li>- LL_AHB1_GRP1_PERIPH_CRYP (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• AHBRSTR DMARST LL_AHB1_GRP1_ReleaseReset</li> <li>• AHBRSTR MIFRST LL_AHB1_GRP1_ReleaseReset</li> <li>• AHBRSTR CRCRST LL_AHB1_GRP1_ReleaseReset</li> <li>• AHBRSTR TSCRST LL_AHB1_GRP1_ReleaseReset</li> <li>• AHBRSTR Rngrst LL_AHB1_GRP1_ReleaseReset</li> <li>• AHBRSTR CRYPRST LL_AHB1_GRP1_ReleaseReset</li> </ul>

### LL\_AHB1\_GRP1\_EnableClockSleep

Function Name	<code>__STATIC_INLINE void LL_AHB1_GRP1_EnableClockSleep (uint32_t Periph)</code>
Function Description	Enable AHB1 peripherals clock during Low Power (Sleep) mode.
Parameters	<ul style="list-style-type: none"> <li><b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_AHB1_GRP1_PERIPH_DMA1</li> <li>- LL_AHB1_GRP1_PERIPH_MIF</li> <li>- LL_AHB1_GRP1_PERIPH_SRAM</li> <li>- LL_AHB1_GRP1_PERIPH_CRC</li> <li>- LL_AHB1_GRP1_PERIPH_TSC (*)</li> <li>- LL_AHB1_GRP1_PERIPH_RNG (*)</li> <li>- LL_AHB1_GRP1_PERIPH_CRYP (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• AHBSMENR DMASMEN LL_AHB1_GRP1_EnableClockSleep</li> <li>• AHBSMENR MIFSMEN LL_AHB1_GRP1_EnableClockSleep</li> <li>• AHBSMENR SRAMSMEN LL_AHB1_GRP1_EnableClockSleep</li> <li>• AHBSMENR CRCSMEN LL_AHB1_GRP1_EnableClockSleep</li> <li>• AHBSMENR TSCSMEN LL_AHB1_GRP1_EnableClockSleep</li> <li>• AHBSMENR RNGSMEN LL_AHB1_GRP1_EnableClockSleep</li> <li>• AHBSMENR CRYPSMEN LL_AHB1_GRP1_EnableClockSleep</li> </ul>

**LL\_AHB1\_GRP1\_DisableClockSleep**

Function Name	<code>__STATIC_INLINE void LL_AHB1_GRP1_DisableClockSleep (uint32_t Periph)</code>
Function Description	Disable AHB1 peripherals clock during Low Power (Sleep) mode.
Parameters	<ul style="list-style-type: none"> <li><b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- <code>LL_AHB1_GRP1_PERIPH_DMA1</code></li> <li>- <code>LL_AHB1_GRP1_PERIPH_MIF</code></li> <li>- <code>LL_AHB1_GRP1_PERIPH_SRAM</code></li> <li>- <code>LL_AHB1_GRP1_PERIPH_CRC</code></li> <li>- <code>LL_AHB1_GRP1_PERIPH_TSC (*)</code></li> <li>- <code>LL_AHB1_GRP1_PERIPH RNG (*)</code></li> <li>- <code>LL_AHB1_GRP1_PERIPH_CRYP (*)</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>- AHBSMENR DMASMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> <li>- AHBSMENR MIFSMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> <li>- AHBSMENR SRAMSMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> <li>- AHBSMENR CRCSMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> <li>- AHBSMENR TSCSMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> <li>- AHBSMENR RNGSMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> <li>- AHBSMENR CRYPSMEN <code>LL_AHB1_GRP1_DisableClockSleep</code></li> </ul>

**LL\_APB1\_GRP1\_EnableClock**

Function Name	<code>__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periph)</code>
Function Description	Enable APB1 peripherals clock.
Parameters	<ul style="list-style-type: none"> <li><b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- <code>LL_APB1_GRP1_PERIPH_TIM2</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_TIM3 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_TIM6 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_TIM7 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_LCD (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_WWDG</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_SPI2 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_USART2</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_LPUART1</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_USART4 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_USART5 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_I2C1</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_I2C2 (*)</code></li> <li>- <code>LL_APB1_GRP1_PERIPH_USB (*)</code></li> </ul> </li> </ul>

- LL\_APB1\_GRP1\_PERIPH\_CRS (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1

**Return values**

Reference Manual to  
LL API cross  
reference:

- **None:**
- APB1ENR TIM2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM6EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM7EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR LCDEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR WWDGEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR SPI2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR USART2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR LPUART1EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR USART4EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR USART5EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR I2C1EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR I2C2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR USBEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR CRSEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR PWREN LL\_APB1\_GRP1\_EnableClock
- APB1ENR DACEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR I2C3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR LPTIM1EN LL\_APB1\_GRP1\_EnableClock

**LL\_APB1\_GRP1\_IsEnabledClock**

Function Name      **STATIC\_INLINE uint32\_t LL\_APB1\_GRP1\_IsEnabledClock  
(uint32\_t Periph)**

Function Description      Check if APB1 peripheral clock is enabled or not.

**Parameters**

- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
  - LL\_APB1\_GRP1\_PERIPH\_TIM2
  - LL\_APB1\_GRP1\_PERIPH\_TIM3 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_TIM6 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_TIM7 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_LCD (\*)
  - LL\_APB1\_GRP1\_PERIPH\_WWDG
  - LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_USART2
  - LL\_APB1\_GRP1\_PERIPH\_LPUART1
  - LL\_APB1\_GRP1\_PERIPH\_USART4 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_USART5 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_I2C1
  - LL\_APB1\_GRP1\_PERIPH\_I2C2 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_USB (\*)
  - LL\_APB1\_GRP1\_PERIPH\_CRS (\*)
  - LL\_APB1\_GRP1\_PERIPH\_PWR
  - LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
  - LL\_APB1\_GRP1\_PERIPH\_I2C3 (\*)

	– LL_APB1_GRP1_PERIPH_LPTIM1
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of Periph (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB1ENR TIM2EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR TIM3EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR TIM6EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR TIM7EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR LCDEN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR WWDGEN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR SPI2EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR USART2EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR LPUART1EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR USART4EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR USART5EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR I2C1EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR I2C2EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR USBEN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR CRSEN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR PWREN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR DACEN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR I2C3EN LL_APB1_GRP1_IsEnabledClock</li> <li>• APB1ENR LPTIM1EN LL_APB1_GRP1_IsEnabledClock</li> </ul>

### LL\_APB1\_GRP1\_DisableClock

Function Name	<b><code>__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periph)</code></b>
Function Description	Disable APB1 peripherals clock.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>– LL_APB1_GRP1_PERIPH_TIM2</li> <li>– LL_APB1_GRP1_PERIPH_TIM3 (*)</li> <li>– LL_APB1_GRP1_PERIPH_TIM6 (*)</li> <li>– LL_APB1_GRP1_PERIPH_TIM7 (*)</li> <li>– LL_APB1_GRP1_PERIPH_LCD (*)</li> <li>– LL_APB1_GRP1_PERIPH_WWDG</li> <li>– LL_APB1_GRP1_PERIPH_SPI2 (*)</li> <li>– LL_APB1_GRP1_PERIPH_USART2</li> <li>– LL_APB1_GRP1_PERIPH_LPUART1</li> <li>– LL_APB1_GRP1_PERIPH_USART4 (*)</li> <li>– LL_APB1_GRP1_PERIPH_USART5 (*)</li> <li>– LL_APB1_GRP1_PERIPH_I2C1</li> <li>– LL_APB1_GRP1_PERIPH_I2C2 (*)</li> <li>– LL_APB1_GRP1_PERIPH_USB (*)</li> <li>– LL_APB1_GRP1_PERIPH_CRS (*)</li> <li>– LL_APB1_GRP1_PERIPH_PWR</li> <li>– LL_APB1_GRP1_PERIPH_DAC1 (*)</li> <li>– LL_APB1_GRP1_PERIPH_I2C3 (*)</li> <li>– LL_APB1_GRP1_PERIPH_LPTIM1</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to	APB1ENR TIM2EN LL_APB1_GRP1_DisableClock

- LL API cross reference:
- APB1ENR TIM3EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR TIM6EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR TIM7EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR LCDEN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR WWDGEN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR SPI2EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR USART2EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR LPUART1EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR USART4EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR USART5EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR I2C1EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR I2C2EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR USBEN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR CRSEN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR PWREN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR DACEN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR I2C3EN LL\_APB1\_GRP1\_DisableClock
  - APB1ENR LPTIM1EN LL\_APB1\_GRP1\_DisableClock

### **LL\_APB1\_GRP1\_ForceReset**

Function Name	<b><code>_STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)</code></b>
Function Description	Force APB1 peripherals reset.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periphs:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB1_GRP1_PERIPH_ALL</li> <li>- LL_APB1_GRP1_PERIPH_TIM2</li> <li>- LL_APB1_GRP1_PERIPH_TIM3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM6 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM7 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LCD (*)</li> <li>- LL_APB1_GRP1_PERIPH_WWDG</li> <li>- LL_APB1_GRP1_PERIPH_SPI2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART2</li> <li>- LL_APB1_GRP1_PERIPH_LPUART1</li> <li>- LL_APB1_GRP1_PERIPH_USART4 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART5 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C1</li> <li>- LL_APB1_GRP1_PERIPH_I2C2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USB (*)</li> <li>- LL_APB1_GRP1_PERIPH_CRS (*)</li> <li>- LL_APB1_GRP1_PERIPH_PWR</li> <li>- LL_APB1_GRP1_PERIPH_DAC1 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LPTIM1</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB1RSTR TIM2RST LL_APB1_GRP1_ForceReset</li> <li>• APB1RSTR TIM3RST LL_APB1_GRP1_ForceReset</li> <li>• APB1RSTR TIM6RST LL_APB1_GRP1_ForceReset</li> <li>• APB1RSTR TIM7RST LL_APB1_GRP1_ForceReset</li> </ul>

- APB1RSTR LCDRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR WWDGRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR SPI2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR USART2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR LPUART1RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR USART4RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR USART5RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR I2C1RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR I2C2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR USBRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR CRSRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR PWRRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR DACRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR I2C3RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR LPTIM1RST LL\_APB1\_GRP1\_ForceReset

### **LL\_APB1\_GRP1\_ReleaseReset**

Function Name	<code>__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periph)</code>
Function Description	Release APB1 peripherals reset.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB1_GRP1_PERIPH_ALL</li> <li>- LL_APB1_GRP1_PERIPH_TIM2</li> <li>- LL_APB1_GRP1_PERIPH_TIM3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM6 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM7 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LCD (*)</li> <li>- LL_APB1_GRP1_PERIPH_WWDG</li> <li>- LL_APB1_GRP1_PERIPH_SPI2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART2</li> <li>- LL_APB1_GRP1_PERIPH_LPUART1</li> <li>- LL_APB1_GRP1_PERIPH_USART4 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART5 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C1</li> <li>- LL_APB1_GRP1_PERIPH_I2C2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USB (*)</li> <li>- LL_APB1_GRP1_PERIPH_CRS (*)</li> <li>- LL_APB1_GRP1_PERIPH_PWR</li> <li>- LL_APB1_GRP1_PERIPH_DAC1 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LPTIM1</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB1RSTR TIM2RST LL_APB1_GRP1_ReleaseReset</li> <li>• APB1RSTR TIM3RST LL_APB1_GRP1_ReleaseReset</li> <li>• APB1RSTR TIM6RST LL_APB1_GRP1_ReleaseReset</li> <li>• APB1RSTR TIM7RST LL_APB1_GRP1_ReleaseReset</li> <li>• APB1RSTR LCDRST LL_APB1_GRP1_ReleaseReset</li> <li>• APB1RSTR WWDGRST LL_APB1_GRP1_ReleaseReset</li> <li>• APB1RSTR SPI2RST LL_APB1_GRP1_ReleaseReset</li> </ul>

- APB1RSTR USART2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR LPUART1RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR USART4RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR USART5RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR I2C1RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR I2C2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR USBRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR CRSRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR PWRRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR DACRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR I2C3RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR LPTIM1RST LL\_APB1\_GRP1\_ReleaseReset

### **LL\_APB1\_GRP1\_EnableClockSleep**

Function Name	<code>__STATIC_INLINE void LL_APB1_GRP1_EnableClockSleep (uint32_t Periph)</code>
Function Description	Enable APB1 peripherals clock during Low Power (Sleep) mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>- LL_APB1_GRP1_PERIPH_TIM2</li> <li>- LL_APB1_GRP1_PERIPH_TIM3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM6 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM7 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LCD (*)</li> <li>- LL_APB1_GRP1_PERIPH_WWDG</li> <li>- LL_APB1_GRP1_PERIPH_SPI2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART2</li> <li>- LL_APB1_GRP1_PERIPH_LPUART1</li> <li>- LL_APB1_GRP1_PERIPH_USART4 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART5 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C1</li> <li>- LL_APB1_GRP1_PERIPH_I2C2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USB (*)</li> <li>- LL_APB1_GRP1_PERIPH_CRS (*)</li> <li>- LL_APB1_GRP1_PERIPH_PWR</li> <li>- LL_APB1_GRP1_PERIPH_DAC1 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LPTIM1</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB1SMENR TIM2SMEN LL_APB1_GRP1_EnableClockSleep</li> <li>• APB1SMENR TIM3SMEN LL_APB1_GRP1_EnableClockSleep</li> <li>• APB1SMENR TIM6SMEN LL_APB1_GRP1_EnableClockSleep</li> <li>• APB1SMENR TIM7SMEN LL_APB1_GRP1_EnableClockSleep</li> <li>• APB1SMENR LCDSMEN LL_APB1_GRP1_EnableClockSleep</li> <li>• APB1SMENR WWDGSMEN</li> </ul>

- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR SPI2SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR USART2SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR LPUART1SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR USART4SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR USART5SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR I2C1SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR I2C2SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR USBSMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR CRSSMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR PWRSMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR DACSMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR I2C3SMEN
- LL\_APB1\_GRP1\_EnableClockSleep
- APB1SMENR LPTIM1SMEN
- LL\_APB1\_GRP1\_EnableClockSleep

### **LL\_APB1\_GRP1\_DisableClockSleep**

Function Name	<b><code>_STATIC_INLINE void LL_APB1_GRP1_DisableClockSleep (uint32_t Periph)</code></b>
Function Description	Disable APB1 peripherals clock during Low Power (Sleep) mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB1_GRP1_PERIPH_TIM2</li> <li>- LL_APB1_GRP1_PERIPH_TIM3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM6 (*)</li> <li>- LL_APB1_GRP1_PERIPH_TIM7 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LCD (*)</li> <li>- LL_APB1_GRP1_PERIPH_WWDG</li> <li>- LL_APB1_GRP1_PERIPH_SPI2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART2</li> <li>- LL_APB1_GRP1_PERIPH_LPUART1</li> <li>- LL_APB1_GRP1_PERIPH_USART4 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USART5 (*)</li> <li>- LL_APB1_GRP1_PERIPH_I2C1</li> <li>- LL_APB1_GRP1_PERIPH_I2C2 (*)</li> <li>- LL_APB1_GRP1_PERIPH_USB (*)</li> <li>- LL_APB1_GRP1_PERIPH_CRS (*)</li> <li>- LL_APB1_GRP1_PERIPH_PWR</li> <li>- LL_APB1_GRP1_PERIPH_DAC1 (*)</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_APB1_GRP1_PERIPH_I2C3 (*)</li> <li>- LL_APB1_GRP1_PERIPH_LPTIM1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB1SMENR TIM2SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR TIM3SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR TIM6SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR TIM7SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR LCDSMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR WWDDGSMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR SPI2SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR USART2SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR LPUART1SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR USART4SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR USART5SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR I2C1SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR I2C2SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR USBSMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR CRSSMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR PWRSMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR DACSMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR I2C3SMEN LL_APB1_GRP1_DisableClockSleep</li> <li>• APB1SMENR LPTIM1SMEN LL_APB1_GRP1_DisableClockSleep</li> </ul>

### LL\_APB2\_GRP1\_EnableClock

Function Name	<code>_STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periph)</code>
Function Description	Enable APB2 peripherals clock.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_SYSCFG</li> <li>- LL_APB2_GRP1_PERIPH_TIM21</li> <li>- LL_APB2_GRP1_PERIPH_TIM22 (*)</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_FW</li> <li>- LL_APB2_GRP1_PERIPH_ADC1</li> <li>- LL_APB2_GRP1_PERIPH_SPI1</li> <li>- LL_APB2_GRP1_PERIPH_USART1 (*)</li> <li>- LL_APB2_GRP1_PERIPH_DBGMCU</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB2ENR SYSCFGEN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR TIM21EN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR TIM22EN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR FWEN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR ADCEN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR SPI1EN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR USART1EN LL_APB2_GRP1_EnableClock</li> <li>• APB2ENR DBGEN LL_APB2_GRP1_EnableClock</li> </ul>

### LL\_APB2\_GRP1\_IsEnabledClock

Function Name	<code>_STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periph)</code>
Function Description	Check if APB2 peripheral clock is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_SYSCFG</li> <li>- LL_APB2_GRP1_PERIPH_TIM21</li> <li>- LL_APB2_GRP1_PERIPH_TIM22 (*)</li> <li>- LL_APB2_GRP1_PERIPH_FW</li> <li>- LL_APB2_GRP1_PERIPH_ADC1</li> <li>- LL_APB2_GRP1_PERIPH_SPI1</li> <li>- LL_APB2_GRP1_PERIPH_USART1 (*)</li> <li>- LL_APB2_GRP1_PERIPH_DBGMCU</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of Periph (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB2ENR SYSCFGEN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR TIM21EN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR TIM22EN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR FWEN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR ADCEN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock</li> <li>• APB2ENR DBGEN LL_APB2_GRP1_IsEnabledClock</li> </ul>

### LL\_APB2\_GRP1\_DisableClock

Function Name	<code>_STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periph)</code>
Function Description	Disable APB2 peripherals clock.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_SYSCFG</li> <li>- LL_APB2_GRP1_PERIPH_TIM21</li> </ul> </li> </ul>

- LL\_APB2\_GRP1\_PERIPH\_TIM22 (\*)
- LL\_APB2\_GRP1\_PERIPH\_FW
- LL\_APB2\_GRP1\_PERIPH\_ADC1
- LL\_APB2\_GRP1\_PERIPH\_SPI1
- LL\_APB2\_GRP1\_PERIPH\_USART1 (\*)
- LL\_APB2\_GRP1\_PERIPH\_DBGMCU

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- APB2ENR SYSCFGEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR TIM21EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR TIM22EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR FWEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR ADCEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SPI1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR USART1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR DBGEN LL\_APB2\_GRP1\_DisableClock

**LL\_APB2\_GRP1\_ForceReset**

<b>Function Name</b>	<code>__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periph)</code>
<b>Function Description</b>	Force APB2 peripherals reset.
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_ALL</li> <li>- LL_APB2_GRP1_PERIPH_SYSCFG</li> <li>- LL_APB2_GRP1_PERIPH_TIM21</li> <li>- LL_APB2_GRP1_PERIPH_TIM22 (*)</li> <li>- LL_APB2_GRP1_PERIPH_ADC1</li> <li>- LL_APB2_GRP1_PERIPH_SPI1</li> <li>- LL_APB2_GRP1_PERIPH_USART1 (*)</li> <li>- LL_APB2_GRP1_PERIPH_DBGMCU</li> </ul> </li> </ul>
<b>Return values</b>	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
<b>Reference Manual to LL API cross reference:</b>	<ul style="list-style-type: none"> <li>• APB2RSTR SYSCFGRST LL_APB2_GRP1_ForceReset</li> <li>• APB2RSTR TIM21RST LL_APB2_GRP1_ForceReset</li> <li>• APB2RSTR TIM22RST LL_APB2_GRP1_ForceReset</li> <li>• APB2RSTR ADCRST LL_APB2_GRP1_ForceReset</li> <li>• APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset</li> <li>• APB2RSTR USART1RST LL_APB2_GRP1_ForceReset</li> <li>• APB2RSTR DBGRST LL_APB2_GRP1_ForceReset</li> </ul>

**LL\_APB2\_GRP1\_ReleaseReset**

<b>Function Name</b>	<code>__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periph)</code>
<b>Function Description</b>	Release APB2 peripherals reset.
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_ALL</li> <li>- LL_APB2_GRP1_PERIPH_SYSCFG</li> </ul> </li> </ul>

- LL\_APB2\_GRP1\_PERIPH\_TIM21
- LL\_APB2\_GRP1\_PERIPH\_TIM22 (\*)
- LL\_APB2\_GRP1\_PERIPH\_ADC1
- LL\_APB2\_GRP1\_PERIPH\_SPI1
- LL\_APB2\_GRP1\_PERIPH\_USART1 (\*)
- LL\_APB2\_GRP1\_PERIPH\_DBGMCU

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- APB2RSTR SYSCFGRST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR TIM21RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR TIM22RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR ADCRST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SPI1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR USART1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR DBGRST LL\_APB2\_GRP1\_ReleaseReset

### **LL\_APB2\_GRP1\_EnableClockSleep**

**Function Name**

**`_STATIC_INLINE void LL_APB2_GRP1_EnableClockSleep  
(uint32_t Periph)`**

**Function Description**

Enable APB2 peripherals clock during Low Power (Sleep) mode.

**Parameters**

- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
  - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
  - LL\_APB2\_GRP1\_PERIPH\_TIM21
  - LL\_APB2\_GRP1\_PERIPH\_TIM22 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC1
  - LL\_APB2\_GRP1\_PERIPH\_SPI1
  - LL\_APB2\_GRP1\_PERIPH\_USART1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DBGMCU

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- APB2SMENR SYSCFGSMEN  
LL\_APB2\_GRP1\_EnableClockSleep
- APB2SMENR TIM21SMEN  
LL\_APB2\_GRP1\_EnableClockSleep
- APB2SMENR TIM22SMEN  
LL\_APB2\_GRP1\_EnableClockSleep
- APB2SMENR ADCSMEN  
LL\_APB2\_GRP1\_EnableClockSleep
- APB2SMENR SPI1SMEN  
LL\_APB2\_GRP1\_EnableClockSleep
- APB2SMENR USART1SMEN  
LL\_APB2\_GRP1\_EnableClockSleep
- APB2SMENR DBGSMEN  
LL\_APB2\_GRP1\_EnableClockSleep

### **LL\_APB2\_GRP1\_DisableClockSleep**

**Function Name**

**`_STATIC_INLINE void LL_APB2_GRP1_DisableClockSleep  
(uint32_t Periph)`**

Function Description	Disable APB2 peripherals clock during Low Power (Sleep) mode.
Parameters	<ul style="list-style-type: none"> <li><b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_APB2_GRP1_PERIPH_SYSCFG</li> <li>- LL_APB2_GRP1_PERIPH_TIM21</li> <li>- LL_APB2_GRP1_PERIPH_TIM22 (*)</li> <li>- LL_APB2_GRP1_PERIPH_ADC1</li> <li>- LL_APB2_GRP1_PERIPH_SPI1</li> <li>- LL_APB2_GRP1_PERIPH_USART1 (*)</li> <li>- LL_APB2_GRP1_PERIPH_DBGMCU</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>APB2SMENR SYSCFGSMEN LL_APB2_GRP1_DisableClockSleep</li> <li>APB2SMENR TIM21SMEN LL_APB2_GRP1_DisableClockSleep</li> <li>APB2SMENR TIM22SMEN LL_APB2_GRP1_DisableClockSleep</li> <li>APB2SMENR ADCSMEN LL_APB2_GRP1_DisableClockSleep</li> <li>APB2SMENR SPI1SMEN LL_APB2_GRP1_DisableClockSleep</li> <li>APB2SMENR USART1SMEN LL_APB2_GRP1_DisableClockSleep</li> <li>APB2SMENR DBGSMEN LL_APB2_GRP1_DisableClockSleep</li> </ul>

### LL\_IOP\_GRP1\_EnableClock

Function Name	<code>__STATIC_INLINE void LL_IOP_GRP1_EnableClock (uint32_t Periph)</code>
Function Description	Enable IOP peripherals clock.
Parameters	<ul style="list-style-type: none"> <li><b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_IOP_GRP1_PERIPH_GPIOA</li> <li>- LL_IOP_GRP1_PERIPH_GPIOB</li> <li>- LL_IOP_GRP1_PERIPH_GPIOC</li> <li>- LL_IOP_GRP1_PERIPH_GPIOD (*)</li> <li>- LL_IOP_GRP1_PERIPH_GPIOE (*)</li> <li>- LL_IOP_GRP1_PERIPH_GPIOH (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IOPENR GPIOAEN LL_IOP_GRP1_EnableClock</li> <li>IOPENR GPIOBEN LL_IOP_GRP1_EnableClock</li> <li>IOPENR GPIOCEN LL_IOP_GRP1_EnableClock</li> <li>IOPENR GPIODEN LL_IOP_GRP1_EnableClock</li> <li>IOPENR GPIOEEN LL_IOP_GRP1_EnableClock</li> <li>IOPENR GPIOHEN LL_IOP_GRP1_EnableClock</li> </ul>

**LL\_IOP\_GRP1\_IsEnabledClock**

Function Name      **`_STATIC_INLINE uint32_t LL_IOP_GRP1_IsEnabledClock  
(uint32_t Periph)`**

Function Description      Check if IOP peripheral clock is enabled or not.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
    - `LL_IOP_GRP1_PERIPH_GPIOA`
    - `LL_IOP_GRP1_PERIPH_GPIOB`
    - `LL_IOP_GRP1_PERIPH_GPIOC`
    - `LL_IOP_GRP1_PERIPH_GPIOD (*)`
    - `LL_IOP_GRP1_PERIPH_GPIOE (*)`
    - `LL_IOP_GRP1_PERIPH_GPIOH (*)`

Return values

- **State:** of Periph (1 or 0).

Reference Manual to  
LL API cross  
reference:

- `IOPENR GPIOAEN LL_IOP_GRP1_IsEnabledClock`
- `IOPENR GPIOBEN LL_IOP_GRP1_IsEnabledClock`
- `IOPENR GPIOCEN LL_IOP_GRP1_IsEnabledClock`
- `IOPENR GPIODEN LL_IOP_GRP1_IsEnabledClock`
- `IOPENR GPIOEEN LL_IOP_GRP1_IsEnabledClock`
- `IOPENR GPIOHEN LL_IOP_GRP1_IsEnabledClock`

**LL\_IOP\_GRP1\_DisableClock**

Function Name      **`_STATIC_INLINE void LL_IOP_GRP1_DisableClock (uint32_t  
Periph)`**

Function Description      Disable IOP peripherals clock.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
    - `LL_IOP_GRP1_PERIPH_GPIOA`
    - `LL_IOP_GRP1_PERIPH_GPIOB`
    - `LL_IOP_GRP1_PERIPH_GPIOC`
    - `LL_IOP_GRP1_PERIPH_GPIOD (*)`
    - `LL_IOP_GRP1_PERIPH_GPIOE (*)`
    - `LL_IOP_GRP1_PERIPH_GPIOH (*)`

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- `IOPENR GPIOAEN LL_IOP_GRP1_DisableClock`
- `IOPENR GPIOBEN LL_IOP_GRP1_DisableClock`
- `IOPENR GPIOCEN LL_IOP_GRP1_DisableClock`
- `IOPENR GPIODEN LL_IOP_GRP1_DisableClock`
- `IOPENR GPIOEEN LL_IOP_GRP1_DisableClock`
- `IOPENR GPIOHEN LL_IOP_GRP1_DisableClock`

**LL\_IOP\_GRP1\_ForceReset**

Function Name      **`_STATIC_INLINE void LL_IOP_GRP1_ForceReset (uint32_t  
Periph)`**

Function Description      Disable IOP peripherals clock.

- Parameters
- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.

- LL\_IOP\_GRP1\_PERIPH\_ALL
- LL\_IOP\_GRP1\_PERIPH\_GPIOA
- LL\_IOP\_GRP1\_PERIPH\_GPIOB
- LL\_IOP\_GRP1\_PERIPH\_GPIOC
- LL\_IOP\_GRP1\_PERIPH\_GPIOD (\*)
- LL\_IOP\_GRP1\_PERIPH\_GPIOE (\*)
- LL\_IOP\_GRP1\_PERIPH\_GPIOH (\*)

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- IOPRSTR GPIOASMEN LL\_IOP\_GRP1\_ForceReset
- IOPRSTR GPIOBSMEN LL\_IOP\_GRP1\_ForceReset
- IOPRSTR GPIOCSMEN LL\_IOP\_GRP1\_ForceReset
- IOPRSTR GPIODSMEN LL\_IOP\_GRP1\_ForceReset
- IOPRSTR GPIOESMEN LL\_IOP\_GRP1\_ForceReset
- IOPRSTR GPIOHSMEN LL\_IOP\_GRP1\_ForceReset

**LL\_IOP\_GRP1\_ReleaseReset****Function Name**

**\_STATIC\_INLINE void LL\_IOP\_GRP1\_ReleaseReset (uint32\_t Periph)**

**Function Description**

Release IOP peripherals reset.

**Parameters**

- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
- LL\_IOP\_GRP1\_PERIPH\_ALL
- LL\_IOP\_GRP1\_PERIPH\_GPIOA
- LL\_IOP\_GRP1\_PERIPH\_GPIOB
- LL\_IOP\_GRP1\_PERIPH\_GPIOC
- LL\_IOP\_GRP1\_PERIPH\_GPIOD (\*)
- LL\_IOP\_GRP1\_PERIPH\_GPIOE (\*)
- LL\_IOP\_GRP1\_PERIPH\_GPIOH (\*)

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- IOPRSTR GPIOASMEN LL\_IOP\_GRP1\_ReleaseReset
- IOPRSTR GPIOBSMEN LL\_IOP\_GRP1\_ReleaseReset
- IOPRSTR GPIOCSMEN LL\_IOP\_GRP1\_ReleaseReset
- IOPRSTR GPIODSMEN LL\_IOP\_GRP1\_ReleaseReset
- IOPRSTR GPIOESMEN LL\_IOP\_GRP1\_ReleaseReset
- IOPRSTR GPIOHSMEN LL\_IOP\_GRP1\_ReleaseReset

**LL\_IOP\_GRP1\_EnableClockSleep****Function Name**

**\_STATIC\_INLINE void LL\_IOP\_GRP1\_EnableClockSleep (uint32\_t Periph)**

**Function Description**

Enable IOP peripherals clock during Low Power (Sleep) mode.

**Parameters**

- **Periph:** This parameter can be a combination of the following values: (\*) value not defined in all devices.
- LL\_IOP\_GRP1\_PERIPH\_GPIOA
- LL\_IOP\_GRP1\_PERIPH\_GPIOB
- LL\_IOP\_GRP1\_PERIPH\_GPIOC
- LL\_IOP\_GRP1\_PERIPH\_GPIOD (\*)
- LL\_IOP\_GRP1\_PERIPH\_GPIOE (\*)

	– LL_IOP_GRP1_PERIPH_GPIOH (*)
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IOPSMENR GPIOARST LL_IOP_GRP1_EnableClockSleep</li> <li>• IOPSMENR GPIOBRST LL_IOP_GRP1_EnableClockSleep</li> <li>• IOPSMENR GPIOCRST LL_IOP_GRP1_EnableClockSleep</li> <li>• IOPSMENR GPIODRST LL_IOP_GRP1_EnableClockSleep</li> <li>• IOPSMENR GPIOERST LL_IOP_GRP1_EnableClockSleep</li> <li>• IOPSMENR GPIOHRST LL_IOP_GRP1_EnableClockSleep</li> </ul>

### LL\_IOP\_GRP1\_DisableClockSleep

Function Name	<b><code>_STATIC_INLINE void LL_IOP_GRP1_DisableClockSleep (uint32_t Periph)</code></b>
Function Description	Disable IOP peripherals clock during Low Power (Sleep) mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices.             <ul style="list-style-type: none"> <li>– LL_IOP_GRP1_PERIPH_GPIOA</li> <li>– LL_IOP_GRP1_PERIPH_GPIOB</li> <li>– LL_IOP_GRP1_PERIPH_GPIOC</li> <li>– LL_IOP_GRP1_PERIPH_GPIOD (*)</li> <li>– LL_IOP_GRP1_PERIPH_GPIOE (*)</li> <li>– LL_IOP_GRP1_PERIPH_GPIOH (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IOPSMENR GPIOARST LL_IOP_GRP1_DisableClockSleep</li> <li>• IOPSMENR GPIOBRST LL_IOP_GRP1_DisableClockSleep</li> <li>• IOPSMENR GPIOCRST LL_IOP_GRP1_DisableClockSleep</li> <li>• IOPSMENR GPIODRST LL_IOP_GRP1_DisableClockSleep</li> <li>• IOPSMENR GPIOERST LL_IOP_GRP1_DisableClockSleep</li> <li>• IOPSMENR GPIOHRST LL_IOP_GRP1_DisableClockSleep</li> </ul>

## 54.2 BUS Firmware driver defines

### 54.2.1 BUS

#### AHB1 GRP1 PERIPH

<code>LL_AHB1_GRP1_PERIPH_ALL</code>	
<code>LL_AHB1_GRP1_PERIPH_DMA1</code>	DMA1 clock enable
<code>LL_AHB1_GRP1_PERIPH_MIF</code>	MIF clock enable
<code>LL_AHB1_GRP1_PERIPH_SRAM</code>	Sleep Mode SRAM clock enable
<code>LL_AHB1_GRP1_PERIPH_CRC</code>	CRC clock enable
<code>LL_AHB1_GRP1_PERIPH_TSC</code>	TSC clock enable
<code>LL_AHB1_GRP1_PERIPH RNG</code>	RNG clock enable
<code>LL_AHB1_GRP1_PERIPH_CRYP</code>	CRYP clock enable

#### APB1 GRP1 PERIPH

<code>LL_APB1_GRP1_PERIPH_ALL</code>	
--------------------------------------	--

LL_APB1_GRP1_PERIPH_TIM2	TIM2 clock enable
LL_APB1_GRP1_PERIPH_TIM3	TIM3 clock enable
LL_APB1_GRP1_PERIPH_TIM6	TIM6 clock enable
LL_APB1_GRP1_PERIPH_TIM7	TIM7 clock enable
LL_APB1_GRP1_PERIPH_LCD	LCD clock enable
LL_APB1_GRP1_PERIPH_WWDG	WWDG clock enable
LL_APB1_GRP1_PERIPH_SPI2	SPI2 clock enable
LL_APB1_GRP1_PERIPH_USART2	USART2 clock enable
LL_APB1_GRP1_PERIPH_LPUART1	LPUART1 clock enable
LL_APB1_GRP1_PERIPH_USART4	USART4 clock enable
LL_APB1_GRP1_PERIPH_USART5	USART5 clock enable
LL_APB1_GRP1_PERIPH_I2C1	I2C1 clock enable
LL_APB1_GRP1_PERIPH_I2C2	I2C2 clock enable
LL_APB1_GRP1_PERIPH_USB	USB clock enable
LL_APB1_GRP1_PERIPH_CRS	CRS clock enable
LL_APB1_GRP1_PERIPH_PWR	PWR clock enable
LL_APB1_GRP1_PERIPH_DAC1	DAC clock enable
LL_APB1_GRP1_PERIPH_I2C3	I2C3 clock enable
LL_APB1_GRP1_PERIPH_LPTIM1	LPTIM1 clock enable
<b>APB2 GRP1 PERIPH</b>	
LL_APB2_GRP1_PERIPH_ALL	
LL_APB2_GRP1_PERIPH_SYSCFG	SYSCFG clock enable
LL_APB2_GRP1_PERIPH_TIM21	TIM21 clock enable
LL_APB2_GRP1_PERIPH_TIM22	TIM22 clock enable
LL_APB2_GRP1_PERIPH_FW	FireWall clock enable
LL_APB2_GRP1_PERIPH_ADC1	ADC1 clock enable
LL_APB2_GRP1_PERIPH_SPI1	SPI1 clock enable
LL_APB2_GRP1_PERIPH_USART1	USART1 clock enable
LL_APB2_GRP1_PERIPH_DBGMCU	DBGMCU clock enable

***IOP GRP1 PERIPH***

LL\_IOP\_GRP1\_PERIPH\_ALL  
LL\_IOP\_GRP1\_PERIPH\_GPIOA GPIO port A control  
LL\_IOP\_GRP1\_PERIPH\_GPIOB GPIO port B control  
LL\_IOP\_GRP1\_PERIPH\_GPIOC GPIO port C control  
LL\_IOP\_GRP1\_PERIPH\_GPIOD GPIO port D control  
LL\_IOP\_GRP1\_PERIPH\_GPIOE GPIO port H control  
LL\_IOP\_GRP1\_PERIPH\_GPIOH GPIO port H control

## 55 LL COMP Generic Driver

### 55.1 COMP Firmware driver registers structures

#### 55.1.1 LL\_COMP\_InitTypeDef

##### Data Fields

- *uint32\_t PowerMode*
- *uint32\_t InputPlus*
- *uint32\_t InputMinus*
- *uint32\_t OutputPolarity*

##### Field Documentation

- ***uint32\_t LL\_COMP\_InitTypeDef::PowerMode***  
Set comparator operating mode to adjust power and speed. This parameter can be a value of **COMP\_LL\_EC\_POWERMODE**This feature can be modified afterwards using unitary function **LL\_COMP\_SetPowerMode()**.
- ***uint32\_t LL\_COMP\_InitTypeDef::InputPlus***  
Set comparator input plus (non-inverting input). This parameter can be a value of **COMP\_LL\_EC\_INPUT\_PLUS**This feature can be modified afterwards using unitary function **LL\_COMP\_SetInputPlus()**.
- ***uint32\_t LL\_COMP\_InitTypeDef::InputMinus***  
Set comparator input minus (inverting input). This parameter can be a value of **COMP\_LL\_EC\_INPUT\_MINUS**This feature can be modified afterwards using unitary function **LL\_COMP\_SetInputMinus()**.
- ***uint32\_t LL\_COMP\_InitTypeDef::OutputPolarity***  
Set comparator output polarity. This parameter can be a value of **COMP\_LL\_EC\_OUTPUT\_POLARITY**This feature can be modified afterwards using unitary function **LL\_COMP\_SetOutputPolarity()**.

### 55.2 COMP Firmware driver API description

#### 55.2.1 Detailed description of functions

##### LL\_COMP\_SetCommonWindowMode

Function Name	<code>__STATIC_INLINE void LL_COMP_SetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON, uint32_t WindowMode)</code>
Function Description	Set window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP<x> and COMP<x+1>).
Parameters	<ul style="list-style-type: none"><li>• <b>COMPxy_COMMON:</b> Comparator common instance (can be set directly from CMSIS definition or by using helper macro <code>__LL_COMP_COMMON_INSTANCE()</code>)</li><li>• <b>WindowMode:</b> This parameter can be one of the following values:<ul style="list-style-type: none"><li>- <code>LL_COMP_WINDOWMODE_DISABLE</code></li></ul></li></ul>

---

– LL\_COMP\_WINDOWMODE\_COMP1\_INPUT\_PLUS\_COMMON

Return values  Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <b>None:</b></li> <li>• COMP1_CSR COMP1WM LL_COMP_SetCommonWindowMode</li> </ul>
--	---

### LL\_COMP\_GetCommonWindowMode

Function Name  Function Description  Parameters  Return values  Reference Manual to LL API cross reference:	<pre style="font-family: monospace; margin: 0;"><b>_STATIC_INLINE uint32_t LL_COMP_GetCommonWindowMode</b> <b>(COMP_Common_TypeDef * COMPxy_COMMON)</b></pre> <p>Get window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP&lt;x&gt; and COMP&lt;x+1&gt;).</p> <ul style="list-style-type: none"> <li>• <b>COMPxy_COMMON:</b> Comparator common instance (can be set directly from CMSIS definition or by using helper macro <code>_LL_COMP_COMMON_INSTANCE()</code>)</li> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_COMP_WINDOWMODE_DISABLE</li> <li>– LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON</li> </ul> </li> <li>• COMP1_CSR COMP1WM LL_COMP_GetCommonWindowMode</li> </ul>
---	---

### LL\_COMP\_SetPowerMode

Function Name  Function Description  Parameters  Return values  Reference Manual to LL API cross reference:	<pre style="font-family: monospace; margin: 0;"><b>_STATIC_INLINE void LL_COMP_SetPowerMode</b> <b>(COMP_TypeDef * COMPx, uint32_t PowerMode)</b></pre> <p>Set comparator instance operating mode to adjust power and speed.</p> <ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> <li>• <b>PowerMode:</b> This parameter can be one of the following values: (1) Available only on COMP instance: COMP2.           <ul style="list-style-type: none"> <li>– LL_COMP_POWERMODE_MEDIUMSPEED (1)</li> <li>– LL_COMP_POWERMODE_ULTRALOWPOWER (1)</li> </ul> </li> <li>• <b>None:</b></li> </ul>
---	--

### LL\_COMP\_GetPowerMode

Function Name  Function Description	<pre style="font-family: monospace; margin: 0;"><b>_STATIC_INLINE uint32_t LL_COMP_GetPowerMode</b> <b>(COMP_TypeDef * COMPx)</b></pre> <p>Get comparator instance operating mode to adjust power and</p>
---	---

	speed.
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: (1) Available only on COMP instance: COMP2.             <ul style="list-style-type: none"> <li>– LL_COMP_POWERMODE_MEDIUMSPEED (1)</li> <li>– LL_COMP_POWERMODE_ULTRALOWPOWER (1)</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Available only on COMP instance: COMP2.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP2_CSR COMP2SPEED LL_COMP_GetPowerMode</li> <li>• </li> </ul>

## LL\_COMP\_ConfigInputs

Function Name	<code>__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)</code>
Function Description	Set comparator inputs minus (inverting) and plus (non-inverting).
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> <li>• <b>InputMinus:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_COMP_INPUT_MINUS_VREFINT</li> <li>– LL_COMP_INPUT_MINUS_IO1</li> <li>– LL_COMP_INPUT_MINUS_DAC1_CH1</li> <li>– LL_COMP_INPUT_MINUS_DAC1_CH2</li> <li>– LL_COMP_INPUT_MINUS_1_4VREFINT</li> <li>– LL_COMP_INPUT_MINUS_1_2VREFINT</li> <li>– LL_COMP_INPUT_MINUS_3_4VREFINT</li> <li>– LL_COMP_INPUT_MINUS_IO2</li> </ul> </li> <li>• <b>InputPlus:</b> This parameter can be one of the following values: (1) Available only on COMP instance: COMP2. (2) Available only on devices STM32L0 category 1.             <ul style="list-style-type: none"> <li>– LL_COMP_INPUT_PLUS_IO1 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO2 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO3 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO4 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO5 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO6 (1)(2)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function shall only be used for COMP2. For setting COMP1 input it is recommended to use LL_COMP_SetInputMinus() Plus (non-inverting) input is not configurable on COMP1. Using this function for COMP1 will corrupt COMP1WM register</li> <li>• On this STM32 serie, specificity if using COMP instance COMP2 with COMP input based on VrefInt (VrefInt or subdivision of VrefInt): scaler bridge is based on VrefInt and requires to enable path from VrefInt (refer to literal SYSCFG_CFGR3_ENBUFLP_VREFINT_COMP).</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• COMP2_CSR COMP2INNSEL LL_COMP_ConfigInputs</li> </ul>

- LL API cross reference:

### LL\_COMP\_SetInputPlus

Function Name	<code>__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)</code>
Function Description	Set comparator input plus (non-inverting).
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> <li>• <b>InputPlus:</b> This parameter can be one of the following values: (1) Available only on COMP instance: COMP2. (2) Available only on devices STM32L0 category 1. <ul style="list-style-type: none"> <li>– LL_COMP_INPUT_PLUS_IO1 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO2 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO3 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO4 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO5 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO6 (1)(2)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Only COMP2 allows to set the input plus (non-inverting). For COMP1 it is always PA1 IO, except when Windows Mode is selected.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP2_CSR COMP2INPSEL LL_COMP_SetInputPlus</li> </ul>

### LL\_COMP\_GetInputPlus

Function Name	<code>__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (COMP_TypeDef * COMPx)</code>
Function Description	Get comparator input plus (non-inverting).
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: (1) Available only on COMP instance: COMP2. (2) Available only on devices STM32L0 category 1. <ul style="list-style-type: none"> <li>– LL_COMP_INPUT_PLUS_IO1 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO2 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO3 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO4 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO5 (1)</li> <li>– LL_COMP_INPUT_PLUS_IO6 (1)(2)</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Only COMP2 allows to set the input plus (non-inverting). For COMP1 it is always PA1 IO, except when Windows Mode is selected.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP2_CSR COMP2INPSEL LL_COMP_GetInputPlus</li> </ul>

**LL\_COMP\_SetInputMinus**

Function Name	<code>_STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)</code>
Function Description	Set comparator input minus (inverting).
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> <li>• <b>InputMinus:</b> This parameter can be one of the following values: (*) Available only on COMP instance: COMP2. <ul style="list-style-type: none"> <li>– LL_COMP_INPUT_MINUS_VREFINT</li> <li>– LL_COMP_INPUT_MINUS_IO1</li> <li>– LL_COMP_INPUT_MINUS_DAC1_CH1</li> <li>– LL_COMP_INPUT_MINUS_DAC1_CH2</li> <li>– LL_COMP_INPUT_MINUS_1_4VREFINT (*)</li> <li>– LL_COMP_INPUT_MINUS_1_2VREFINT (*)</li> <li>– LL_COMP_INPUT_MINUS_3_4VREFINT (*)</li> <li>– LL_COMP_INPUT_MINUS_IO2 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.</li> <li>• On this STM32 serie, specificity if using COMP instance COMP2 with COMP input based on VrefInt (VrefInt or subdivision of VrefInt): scaler bridge is based on VrefInt and requires to enable path from VrefInt (refer to literal SYSCFG_CFGR3_ENBUFLP_VREFINT_COMP).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1INNSEL LL_COMP_SetInputMinus</li> <li>• COMP2_CSR COMP2INNSEL LL_COMP_SetInputMinus</li> </ul>

**LL\_COMP\_GetInputMinus**

Function Name	<code>_STATIC_INLINE uint32_t LL_COMP_GetInputMinus (COMP_TypeDef * COMPx)</code>
Function Description	Get comparator input minus (inverting).
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: (*) Available only on COMP instance: COMP2. <ul style="list-style-type: none"> <li>– LL_COMP_INPUT_MINUS_VREFINT</li> <li>– LL_COMP_INPUT_MINUS_IO1</li> <li>– LL_COMP_INPUT_MINUS_DAC1_CH1</li> <li>– LL_COMP_INPUT_MINUS_DAC1_CH2</li> <li>– LL_COMP_INPUT_MINUS_1_4VREFINT (*)</li> <li>– LL_COMP_INPUT_MINUS_1_2VREFINT (*)</li> <li>– LL_COMP_INPUT_MINUS_3_4VREFINT (*)</li> <li>– LL_COMP_INPUT_MINUS_IO2 (*)</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1INNSEL LL_COMP_GetInputMinus</li> </ul>

- LL API cross reference:
- COMP2\_CSR COMP2INNSEL LL\_COMP\_GetInputMinus

### **LL\_COMP\_SetOutputLPTIM**

Function Name	<b><code>_STATIC_INLINE void LL_COMP_SetOutputLPTIM (COMP_TypeDef * COMPx, uint32_t OutputLptim)</code></b>
Function Description	Set comparator output LPTIM.
Parameters	<ul style="list-style-type: none"> <li><b>COMPx:</b> Comparator instance</li> <li><b>OutputLptim:</b> This parameter can be one of the following values: (*) Available only on COMP instance: COMP1.             <ul style="list-style-type: none"> <li>- LL_COMP_OUTPUT_LPTIM1_IN1_COMP1 (*)</li> <li>- LL_COMP_OUTPUT_LPTIM1_IN1_COMP2 (**)</li> <li>- LL_COMP_OUTPUT_LPTIM1_IN2_COMP2 (**)</li> </ul> </li> <li>(**) Available only on COMP instance: COMP2.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>COMP1_CSR COMP1LPTIMIN1 LL_COMP_SetOutputLPTIM</li> <li>COMP2_CSR COMP2LPTIMIN1 LL_COMP_SetOutputLPTIM</li> <li>COMP2_CSR COMP2LPTIMIN2 LL_COMP_SetOutputLPTIM</li> </ul>

### **LL\_COMP\_GetOutputLPTIM**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_COMP_GetOutputLPTIM (COMP_TypeDef * COMPx)</code></b>
Function Description	Get comparator output LPTIM.
Parameters	<ul style="list-style-type: none"> <li><b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: (*) Available only on COMP instance: COMP1.             <ul style="list-style-type: none"> <li>- LL_COMP_OUTPUT_LPTIM1_IN1_COMP1 (*)</li> <li>- LL_COMP_OUTPUT_LPTIM1_IN1_COMP2 (**)</li> <li>- LL_COMP_OUTPUT_LPTIM1_IN2_COMP2 (**)</li> </ul> </li> <li>(**) Available only on COMP instance: COMP2.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>COMP1_CSR COMP1LPTIMIN1 LL_COMP_GetOutputLPTIM</li> <li>COMP2_CSR COMP2LPTIMIN1 LL_COMP_GetOutputLPTIM</li> <li>COMP2_CSR COMP2LPTIMIN2 LL_COMP_GetOutputLPTIM</li> </ul>

### **LL\_COMP\_SetOutputPolarity**

Function Name	<b><code>_STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)</code></b>
Function Description	Set comparator instance output polarity.
Parameters	<ul style="list-style-type: none"> <li><b>COMPx:</b> Comparator instance</li> <li><b>OutputPolarity:</b> This parameter can be one of the following</li> </ul>

	<p>values:</p> <ul style="list-style-type: none"> <li>- LL_COMP_OUTPUTPOL_NONINVERTED</li> <li>- LL_COMP_OUTPUTPOL_INVERTED</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP COMP1POLARITY LL_COMP_SetOutputPolarity</li> </ul>

### **LL\_COMP\_GetOutputPolarity**

Function Name	<b><u>__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (COMP_TypeDef * COMPx)</u></b>
Function Description	Get comparator instance output polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_COMP_OUTPUTPOL_NONINVERTED</li> <li>- LL_COMP_OUTPUTPOL_INVERTED</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP COMP1POLARITY LL_COMP_GetOutputPolarity</li> </ul>

### **LL\_COMP\_Enable**

Function Name	<b><u>__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)</u></b>
Function Description	Enable comparator instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After enable from off state, comparator requires a delay to reach propagation delay specification. Refer to device datasheet, parameter "tSTART".</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1EN LL_COMP_Enable</li> <li>• COMP2_CSR COMP2EN LL_COMP_Enable</li> </ul>

### **LL\_COMP\_Disable**

Function Name	<b><u>__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)</u></b>
Function Description	Disable comparator instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1EN LL_COMP_Disable</li> <li>• COMP2_CSR COMP2EN LL_COMP_Disable</li> </ul>

**LL\_COMP\_IsEnabled**

Function Name	<code>_STATIC_INLINE uint32_t LL_COMP_IsEnabled (COMP_TypeDef * COMPx)</code>
Function Description	Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1EN LL_COMP_IsEnabled</li> <li>• COMP2_CSR COMP2EN LL_COMP_IsEnabled</li> </ul>

**LL\_COMP\_Lock**

Function Name	<code>_STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)</code>
Function Description	Lock comparator instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once locked, comparator configuration can be accessed in read-only.</li> <li>• The only way to unlock the comparator is a device hardware reset.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1LOCK LL_COMP_Lock</li> <li>• COMP2_CSR COMP2LOCK LL_COMP_Lock</li> </ul>

**LL\_COMP\_IsLocked**

Function Name	<code>_STATIC_INLINE uint32_t LL_COMP_IsLocked (COMP_TypeDef * COMPx)</code>
Function Description	Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).
Parameters	<ul style="list-style-type: none"> <li>• <b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once locked, comparator configuration can be accessed in read-only.</li> <li>• The only way to unlock the comparator is a device hardware reset.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• COMP1_CSR COMP1LOCK LL_COMP_IsLocked</li> <li>• COMP2_CSR COMP2LOCK LL_COMP_IsLocked</li> </ul>

**LL\_COMP\_ReadOutputLevel**

Function Name	<code>_STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (COMP_TypeDef * COMPx)</code>
---------------	---

Function Description	Read comparator instance output level.
Parameters	<ul style="list-style-type: none"> <li><b>COMPx:</b> Comparator instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_COMP_OUTPUT_LEVEL_LOW</li> <li>– LL_COMP_OUTPUT_LEVEL_HIGH</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>The comparator output level depends on the selected polarity (Refer to function LL_COMP_SetOutputPolarity()). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>COMP1_CSR COMP1VALUE LL_COMP_ReadOutputLevel</li> <li>COMP2_CSR COMP2VALUE LL_COMP_ReadOutputLevel</li> </ul>

### LL\_COMP\_DeInit

Function Name	<b>ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)</b>
Function Description	De-initialize registers of the selected COMP instance to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>COMPx:</b> COMP instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:             <ul style="list-style-type: none"> <li>– SUCCESS: COMP registers are de-initialized</li> <li>– ERROR: COMP registers are not de-initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.</li> </ul>

### LL\_COMP\_Init

Function Name	<b>ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, LL_COMP_InitTypeDef * COMP_InitStruct)</b>
Function Description	Initialize some features of COMP instance.
Parameters	<ul style="list-style-type: none"> <li><b>COMPx:</b> COMP instance</li> <li><b>COMP_InitStruct:</b> Pointer to a LL_COMP_InitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:             <ul style="list-style-type: none"> <li>– SUCCESS: COMP registers are initialized</li> <li>– ERROR: COMP registers are not initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.</li> </ul>

**LL\_COMP\_StructInit**

Function Name	<code>void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)</code>
Function Description	Set each LL_COMP_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>COMP_InitStruct:</b> pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**55.3 COMP Firmware driver defines****55.3.1 COMP*****Comparator common modes - Window mode***

<code>LL_COMP_WINDOWMODE_DISABLE</code>	Window mode disable: Comparators 1 and 2 are independent
<code>LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON</code>	Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

***Definitions of COMP hardware constraints delays***

<code>LL_COMP_DELAY_STARTUP_US</code>	Delay for COMP startup time
<code>LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US</code>	Delay for COMP voltage scaler stabilization time

***Comparator inputs - Input minus (input inverting) selection***

<code>LL_COMP_INPUT_MINUS_1_4VREFINT</code>	Comparator input minus connected to 1/4 VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function)
<code>LL_COMP_INPUT_MINUS_1_2VREFINT</code>	Comparator input minus connected to 1/2 VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function)
<code>LL_COMP_INPUT_MINUS_3_4VREFINT</code>	Comparator input minus connected to 3/4 VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function)
<code>LL_COMP_INPUT_MINUS_VREFINT</code>	Comparator input minus connected to VrefInt (specifity of COMP2 related to path to enable via SYSCFG: refer to comment in function)
<code>LL_COMP_INPUT_MINUS_DAC1_CH1</code>	Comparator input minus connected to DAC1

LL_COMP_INPUT_MINUS_DAC1_CH2	channel 1 (DAC_OUT1) Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)
LL_COMP_INPUT_MINUS_IO1	Comparator input minus connected to IO1 (pin PA0 for COMP1, pin PA2 for COMP2)
LL_COMP_INPUT_MINUS_IO2	Comparator input minus connected to IO2 (pin PB3 for COMP2) (specific to COMP instance: COMP2)

***Comparator inputs - Input plus (input non-inverting) selection***

LL_COMP_INPUT_PLUS_IO1	Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA3 for COMP2)
LL_COMP_INPUT_PLUS_IO2	Comparator input plus connected to IO2 (pin PB4 for COMP2) (specific to COMP instance: COMP2)
LL_COMP_INPUT_PLUS_IO3	Comparator input plus connected to IO3 (pin PA5 for COMP2) (specific to COMP instance: COMP2)
LL_COMP_INPUT_PLUS_IO4	Comparator input plus connected to IO4 (pin PB6 for COMP2) (specific to COMP instance: COMP2)
LL_COMP_INPUT_PLUS_IO5	Comparator input plus connected to IO5 (pin PB7 for COMP2) (specific to COMP instance: COMP2)

***Comparator output - Output level***

LL_COMP_OUTPUT_LEVEL_LOW	Comparator output level low (if the polarity is not inverted, otherwise to be complemented)
LL_COMP_OUTPUT_LEVEL_HIGH	Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

***Comparator output - Output polarity***

LL_COMP_OUTPUTPOL_NONINVERTED	COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input
LL_COMP_OUTPUTPOL_INVERTED	COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

***Comparator output - Output selection specific to LPTIM peripheral***

LL_COMP_OUTPUT_LPTIM1_IN1_COMP1	COMP output connected to TIM2 input capture 4
LL_COMP_OUTPUT_LPTIM1_IN1_COMP2	COMP output connected to TIM2 input capture 4
LL_COMP_OUTPUT_LPTIM1_IN2_COMP2	COMP output connected to TIM2 input capture 4

***Comparator modes - Power mode***

LL_COMP_POWERMODE_ULTRALOWPOWER	COMP power mode to low speed (specific to COMP instance: COMP2)
LL_COMP_POWERMODE_MEDIUMSPEED	COMP power mode to fast speed

***COMP helper macro*****\_LL\_COMP\_COMMON\_INSTANCE**   **Description:**

- Helper macro to select the COMP common instance to which is belonging the selected COMP instance.

**Parameters:**

- \_\_COMPx\_\_: COMP instance

**Return value:**

- COMP: common instance or value "0" if there is no COMP common instance.

**Notes:**

- COMP common register instance can be used to set parameters common to several COMP instances. Refer to functions having argument "COMPxy\_COMMON" as parameter.

***Common write and read registers macro*****LL\_COMP\_WriteReg**   **Description:**

- Write a value in COMP register.

**Parameters:**

- \_\_INSTANCE\_\_: comparator instance
- \_\_REG\_\_: Register to be written
- \_\_VALUE\_\_: Value to be written in the register

**Return value:**

- None

**LL\_COMP\_ReadReg**   **Description:**

- Read a value in COMP register.

**Parameters:**

- \_\_INSTANCE\_\_: comparator instance
- \_\_REG\_\_: Register to be read

**Return value:**

- Register: value

## 56 LL CORTEX Generic Driver

### 56.1 CORTEX Firmware driver API description

#### 56.1.1 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	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• STK_CTRL TICKINT LL_SYSTICK_EnableIT</li> </ul>

### **LL\_SYSTICK\_DisableIT**

Function Name      **\_\_STATIC\_INLINE void LL\_SYSTICK\_DisableIT (void )**

Function Description      Disable SysTick exception request.

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• STK_CTRL TICKINT LL_SYSTICK_DisableIT</li> </ul>

### **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	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT</li> </ul>

### **LL\_LPM\_EnableSleep**

Function Name      **\_\_STATIC\_INLINE void LL\_LPM\_EnableSleep (void )**

Function Description      Processor uses sleep as its low power mode.

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SCB_SCR SLEEPDEEP LL_LPM_EnableSleep</li> </ul>

### **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	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep</li> </ul>

### **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	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit</li> </ul>

### **LL\_LPM\_DisableSleepOnExit**

Function Name	<code>__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void )</code>
Function Description	Do not sleep when returning to Thread mode.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit</li> </ul>

### **LL\_LPM\_EnableEventOnPend**

Function Name	<code>__STATIC_INLINE void LL_LPM_EnableEventOnPend (void )</code>
Function Description	Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend</li> </ul>

### **LL\_LPM\_DisableEventOnPend**

Function Name	<code>__STATIC_INLINE void LL_LPM_DisableEventOnPend (void )</code>
Function Description	Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend</li> </ul>

### **LL\_CPUID\_GetImplementer**

Function Name	<code>__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void )</code>
Function Description	Get Implementer code.
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> should be equal to 0x41 for ARM</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer</li> </ul>

**LL\_CPUID\_GetVariant**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void )</code></b>
Function Description	Get Variant number (The r value in the rnnp product revision identifier)
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between 0 and 255 (0x0: revision 0)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SCB_CPUID VARIANT LL_CPUID_GetVariant</li> </ul>

**LL\_CPUID\_GetArchitecture**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void )</code></b>
Function Description	Get Architecture number.
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> should be equal to 0xC for Cortex-M0+ devices</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture</li> </ul>

**LL\_CPUID\_GetParNo**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void )</code></b>
Function Description	Get Part number.
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> should be equal to 0xC60 for Cortex-M0+</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SCB_CPUID PARTNO LL_CPUID_GetParNo</li> </ul>

**LL\_CPUID\_GetRevision**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void )</code></b>
Function Description	Get Revision number (The p value in the rnnp product revision identifier, indicates patch release)
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between 0 and 255 (0x1: patch 1)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SCB_CPUID REVISION LL_CPUID_GetRevision</li> </ul>

**LL\_MPU\_Enable**

Function Name	<b><code>__STATIC_INLINE void LL_MPU_Enable (uint32_t Options)</code></b>
Function Description	Enable MPU with input options.
Parameters	<ul style="list-style-type: none"> <li>• <b>Options:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_MPU_CTRL_HFNMI_PRIVDEF_NONE</code></li> <li>- <code>LL_MPU_CTRL_HARDFAULT_NMI</code></li> <li>- <code>LL_MPU_CTRL_PRIVILEGED_DEFAULT</code></li> <li>- <code>LL_MPU_CTRL_HFNMI_PRIVDEF</code></li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>MPU_CTRL ENABLE LL_MPU_Enable</li> </ul>

### **LL\_MPU\_Disable**

Function Name	<b><u>__STATIC_INLINE void LL_MPU_Disable (void )</u></b>
Function Description	Disable MPU.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>MPU_CTRL ENABLE LL_MPU_Disable</li> </ul>

### **LL\_MPU\_IsEnabled**

Function Name	<b><u>__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void )</u></b>
Function Description	Check if MPU is enabled or not.
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>MPU_CTRL ENABLE LL_MPU_IsEnabled</li> </ul>

### **LL\_MPU\_EnableRegion**

Function Name	<b><u>__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)</u></b>
Function Description	Enable a MPU region.
Parameters	<ul style="list-style-type: none"> <li><b>Region:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_MPU_REGION_NUMBER0</li> <li>- LL_MPU_REGION_NUMBER1</li> <li>- LL_MPU_REGION_NUMBER2</li> <li>- LL_MPU_REGION_NUMBER3</li> <li>- LL_MPU_REGION_NUMBER4</li> <li>- LL_MPU_REGION_NUMBER5</li> <li>- LL_MPU_REGION_NUMBER6</li> <li>- LL_MPU_REGION_NUMBER7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>MPU_RASR ENABLE LL_MPU_EnableRegion</li> </ul>

### **LL\_MPU\_ConfigRegion**

Function Name	<b><u>__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t SubRegionDisable, uint32_t Address, uint32_t Attributes)</u></b>
---------------	--

Function Description	Configure and enable a region.
Parameters	<ul style="list-style-type: none"> <li>• <b>Region:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_MPUM_REGION_NUMBER0</li> <li>- LL_MPUM_REGION_NUMBER1</li> <li>- LL_MPUM_REGION_NUMBER2</li> <li>- LL_MPUM_REGION_NUMBER3</li> <li>- LL_MPUM_REGION_NUMBER4</li> <li>- LL_MPUM_REGION_NUMBER5</li> <li>- LL_MPUM_REGION_NUMBER6</li> <li>- LL_MPUM_REGION_NUMBER7</li> </ul> </li> <li>• <b>Address:</b> Value of region base address</li> <li>• <b>SubRegionDisable:</b> Sub-region disable value between Min_Data = 0x00 and Max_Data = 0xFF</li> <li>• <b>Attributes:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_MPUM_REGION_SIZE_32B or LL_MPUM_REGION_SIZE_64B or LL_MPUM_REGION_SIZE_128B or LL_MPUM_REGION_SIZE_256B or LL_MPUM_REGION_SIZE_512B or LL_MPUM_REGION_SIZE_1KB or LL_MPUM_REGION_SIZE_2KB or LL_MPUM_REGION_SIZE_4KB or LL_MPUM_REGION_SIZE_8KB or LL_MPUM_REGION_SIZE_16KB or LL_MPUM_REGION_SIZE_32KB or LL_MPUM_REGION_SIZE_64KB or LL_MPUM_REGION_SIZE_128KB or LL_MPUM_REGION_SIZE_256KB or LL_MPUM_REGION_SIZE_512KB or LL_MPUM_REGION_SIZE_1MB or LL_MPUM_REGION_SIZE_2MB or LL_MPUM_REGION_SIZE_4MB or LL_MPUM_REGION_SIZE_8MB or LL_MPUM_REGION_SIZE_16MB or LL_MPUM_REGION_SIZE_32MB or LL_MPUM_REGION_SIZE_64MB or LL_MPUM_REGION_SIZE_128MB or LL_MPUM_REGION_SIZE_256MB or LL_MPUM_REGION_SIZE_512MB or LL_MPUM_REGION_SIZE_1GB or LL_MPUM_REGION_SIZE_2GB or LL_MPUM_REGION_SIZE_4GB</li> <li>- LL_MPUM_REGION_NO_ACCESS or LL_MPUM_REGION_PRIV_RW or LL_MPUM_REGION_PRIV_RW_URO or LL_MPUM_REGION_FULL_ACCESS or LL_MPUM_REGION_PRIV_RO or LL_MPUM_REGION_PRIV_RO_URO</li> <li>- LL_MPUM_TEX_LEVEL0 or LL_MPUM_TEX_LEVEL1 or LL_MPUM_TEX_LEVEL2 or LL_MPUM_TEX_LEVEL4</li> <li>- LL_MPUM_INSTRUCTION_ACCESS_ENABLE or LL_MPUM_INSTRUCTION_ACCESS_DISABLE</li> <li>- LL_MPUM_ACCESS_SHAREABLE or</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_MPU_ACCESS_NOT_SHAREABLE</li> <li>- LL_MPU_ACCESS_CACHEABLE or LL_MPU_ACCESS_NOT_CACHEABLE</li> <li>- LL_MPU_ACCESS_BUFFERABLE or LL_MPU_ACCESS_NOT_BUFFERABLE</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• MPU_RNR REGION LL_MPU_ConfigRegion</li> <li>• MPU_RBAR REGION LL_MPU_ConfigRegion</li> <li>• MPU_RBAR ADDR LL_MPU_ConfigRegion</li> <li>• MPU_RASR XN LL_MPU_ConfigRegion</li> <li>• MPU_RASR AP LL_MPU_ConfigRegion</li> <li>• MPU_RASR S LL_MPU_ConfigRegion</li> <li>• MPU_RASR C LL_MPU_ConfigRegion</li> <li>• MPU_RASR B LL_MPU_ConfigRegion</li> <li>• MPU_RASR SIZE LL_MPU_ConfigRegion</li> </ul>

### LL\_MPU\_DisableRegion

Function Name	<code>__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)</code>
Function Description	Disable a region.
Parameters	<ul style="list-style-type: none"> <li>• <b>Region:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_MPU_REGION_NUMBER0</li> <li>- LL_MPU_REGION_NUMBER1</li> <li>- LL_MPU_REGION_NUMBER2</li> <li>- LL_MPU_REGION_NUMBER3</li> <li>- LL_MPU_REGION_NUMBER4</li> <li>- LL_MPU_REGION_NUMBER5</li> <li>- LL_MPU_REGION_NUMBER6</li> <li>- LL_MPU_REGION_NUMBER7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• MPU_RNR REGION LL_MPU_DisableRegion</li> <li>• MPU_RASR ENABLE LL_MPU_DisableRegion</li> </ul>

## 56.2 CORTEX Firmware driver defines

### 56.2.1 CORTEX

#### *MPU Bufferable Access*

<code>LL_MPU_ACCESS_BUFFERABLE</code>	Bufferable memory attribute
<code>LL_MPU_ACCESS_NOT_BUFFERABLE</code>	Not Bufferable memory attribute

#### *MPU Cacheable Access*

<code>LL_MPU_ACCESS_CACHEABLE</code>	Cacheable memory attribute
<code>LL_MPU_ACCESS_NOT_CACHEABLE</code>	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_MPUCtrl_HFNMI_PRIVDEF_NONE` Disable NMI and privileged SW access

`LL_MPUCtrl_HARDFAULT_NMI` Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers

`LL_MPUCtrl_PRIVILEGED_DEFAULT` Enable privileged software access to default memory map

`LL_MPUCtrl_HFNMI_PRIVDEF` Enable NMI and privileged SW access

#### ***MPU Instruction Access***

`LL_MPUCtrl_INSTRUCTION_ACCESS_ENABLE` Instruction fetches enabled

`LL_MPUCtrl_INSTRUCTION_ACCESS_DISABLE` Instruction fetches disabled

#### ***MPU Region Number***

`LL_MPURegion_Number0` REGION Number 0

`LL_MPURegion_Number1` REGION Number 1

`LL_MPURegion_Number2` REGION Number 2

`LL_MPURegion_Number3` REGION Number 3

`LL_MPURegion_Number4` REGION Number 4

`LL_MPURegion_Number5` REGION Number 5

`LL_MPURegion_Number6` REGION Number 6

`LL_MPURegion_Number7` REGION Number 7

#### ***MPU Region Privileges***

`LL_MPURegion_No_Access` No access

`LL_MPURegion_Priv_RW` RW privileged (privileged access only)

`LL_MPURegion_Priv_RW_Uro` RW privileged - RO user (Write in a user program generates a fault)

`LL_MPURegion_Full_Access` RW privileged & user (Full access)

`LL_MPURegion_Priv_RO` RO privileged (privileged read only)

`LL_MPURegion_Priv_RO_Uro` RO privileged & user (read only)

#### ***MPU Region Size***

`LL_MPURegion_Size_32B` 32B Size of the MPU protection region

`LL_MPURegion_Size_64B` 64B Size of the MPU protection region

`LL_MPURegion_Size_128B` 128B Size of the MPU protection region

`LL_MPURegion_Size_256B` 256B Size of the MPU protection region

`LL_MPURegion_Size_512B` 512B Size of the MPU protection region

`LL_MPURegion_Size_1KB` 1KB Size of the MPU protection region

`LL_MPURegion_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

## 57 LL CRC Generic Driver

### 57.1 CRC Firmware driver API description

#### 57.1.1 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:**

Notes      • If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC\_INIT register, otherwise, reset Data Register to its default value.

Reference Manual to  
LL API cross  
reference:      • CR RESET LL\_CRC\_ResetCRCCalculationUnit

##### **LL\_CRC\_SetPolynomialSize**

Function Name      **\_STATIC\_INLINE void LL\_CRC\_SetPolynomialSize (CRC\_TypeDef \* CRCx, uint32\_t PolySize)**

Function Description      Configure size of the polynomial.

Parameters      • **CRCx:** CRC Instance

Parameters      • **PolySize:** This parameter can be one of the following values:  
 – LL\_CRC\_POLYLENGTH\_32B  
 – LL\_CRC\_POLYLENGTH\_16B  
 – LL\_CRC\_POLYLENGTH\_8B  
 – LL\_CRC\_POLYLENGTH\_7B

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:      • CR POLYSIZE LL\_CRC\_SetPolynomialSize

##### **LL\_CRC\_GetPolynomialSize**

Function Name      **\_STATIC\_INLINE uint32\_t LL\_CRC\_GetPolynomialSize (CRC\_TypeDef \* CRCx)**

Function Description      Return size of the polynomial.

Parameters      • **CRCx:** CRC Instance

Return values      • **Returned:** value can be one of the following values:  
 – LL\_CRC\_POLYLENGTH\_32B  
 – LL\_CRC\_POLYLENGTH\_16B

Reference Manual to  
LL API cross  
reference:

- LL\_CRC\_POLYLENGTH\_8B
- LL\_CRC\_POLYLENGTH\_7B
- CR POLYSIZE LL\_CRC\_GetPolynomialSize

### **LL\_CRC\_SetInputDataReverseMode**

Function Name      **STATIC\_INLINE void LL\_CRC\_SetInputDataReverseMode (CRC\_TypeDef \* CRCx, uint32\_t ReverseMode)**

Function Description      Configure the reversal of the bit order of the input data.

Parameters     
 

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL\_CRC\_INDATA\_REVERSE\_NONE
  - LL\_CRC\_INDATA\_REVERSE\_BYTE
  - LL\_CRC\_INDATA\_REVERSE\_HALFWORD
  - LL\_CRC\_INDATA\_REVERSE\_WORD

Return values     
 

- **None:**

Reference Manual to  
LL API cross  
reference:

- CR REV\_IN LL\_CRC\_SetInputDataReverseMode

Function Name      **STATIC\_INLINE uint32\_t LL\_CRC\_GetInputDataReverseMode (CRC\_TypeDef \* CRCx)**

Function Description      Return type of reversal for input data bit order.

Parameters     
 

- **CRCx:** CRC Instance

Return values     
 

- **Returned:** value can be one of the following values:
  - LL\_CRC\_INDATA\_REVERSE\_NONE
  - LL\_CRC\_INDATA\_REVERSE\_BYTE
  - LL\_CRC\_INDATA\_REVERSE\_HALFWORD
  - LL\_CRC\_INDATA\_REVERSE\_WORD

Reference Manual to  
LL API cross  
reference:

- CR REV\_IN LL\_CRC\_GetInputDataReverseMode

### **LL\_CRC\_SetOutputDataReverseMode**

Function Name      **STATIC\_INLINE void LL\_CRC\_SetOutputDataReverseMode (CRC\_TypeDef \* CRCx, uint32\_t ReverseMode)**

Function Description      Configure the reversal of the bit order of the Output data.

Parameters     
 

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL\_CRC\_OUTDATA\_REVERSE\_NONE
  - LL\_CRC\_OUTDATA\_REVERSE\_BIT

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR REV_OUT LL_CRC_SetOutputDataReverseMode</li> </ul>

### LL\_CRC\_GetOutputDataReverseMode

Function Name	<code>__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (CRC_TypeDef * CRCx)</code>
Function Description	Configure the reversal of the bit order of the Output data.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>LL_CRC_OUTDATA_REVERSE_NONE</li> <li>LL_CRC_OUTDATA_REVERSE_BIT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR REV_OUT LL_CRC_GetOutputDataReverseMode</li> </ul>

### LL\_CRC\_SetInitialData

Function Name	<code>__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)</code>
Function Description	Initialize the Programmable initial CRC value.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> <li><b>InitCrc:</b> Value to be programmed in Programmable initial CRC value register</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>If the CRC size is less than 32 bits, the least significant bits are used to write the correct value</li> <li>LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>INIT INIT LL_CRC_SetInitialData</li> </ul>

### LL\_CRC\_GetInitialData

Function Name	<code>__STATIC_INLINE uint32_t LL_CRC_GetInitialData (CRC_TypeDef * CRCx)</code>
Function Description	Return current Initial CRC value.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> programmed in Programmable initial CRC value register</li> </ul>
Notes	<ul style="list-style-type: none"> <li>If the CRC size is less than 32 bits, the least significant bits are used to read the correct value</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>INIT INIT LL_CRC_GetInitialData</li> </ul>

LL API cross  
reference:

### **LL\_CRC\_SetPolynomialCoef**

Function Name	<b><code>_STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)</code></b>
Function Description	Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).
Parameters	<ul style="list-style-type: none"> <li>• <b>CRCx:</b> CRC Instance</li> <li>• <b>PolynomCoef:</b> Value to be programmed in Programmable Polynomial value register</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter.</li> <li>• Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, <math>X^7 + X^6 + X^5 + X^2 + 1</math> is written 0x65</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• POL POL LL_CRC_SetPolynomialCoef</li> </ul>

### **LL\_CRC\_GetPolynomialCoef**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (CRC_TypeDef * CRCx)</code></b>
Function Description	Return current Programmable polynomial value.
Parameters	<ul style="list-style-type: none"> <li>• <b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> programmed in Programmable Polynomial value register</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, <math>X^7 + X^6 + X^5 + X^2 + 1</math> is written 0x65</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• POL POL LL_CRC_GetPolynomialCoef</li> </ul>

### **LL\_CRC\_FeedData32**

Function Name	<b><code>_STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)</code></b>
Function Description	Write given 32-bit data to the CRC calculator.
Parameters	<ul style="list-style-type: none"> <li>• <b>CRCx:</b> CRC Instance</li> <li>• <b>InData:</b> value to be provided to CRC calculator between Min_Data=0 and Max_Data=0xFFFFFFFF</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DR LL_CRC_FeedData32</li> </ul>

### LL\_CRC\_FeedData16

Function Name	<code>__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)</code>
Function Description	Write given 16-bit data to the CRC calculator.
Parameters	<ul style="list-style-type: none"> <li>• <b>CRCx:</b> CRC Instance</li> <li>• <b>InData:</b> 16 bit value to be provided to CRC calculator between Min_Data=0 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DR LL_CRC_FeedData16</li> </ul>

### LL\_CRC\_FeedData8

Function Name	<code>__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)</code>
Function Description	Write given 8-bit data to the CRC calculator.
Parameters	<ul style="list-style-type: none"> <li>• <b>CRCx:</b> CRC Instance</li> <li>• <b>InData:</b> 8 bit value to be provided to CRC calculator between Min_Data=0 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DR LL_CRC_FeedData8</li> </ul>

### LL\_CRC\_ReadData32

Function Name	<code>__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)</code>
Function Description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none"> <li>• <b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Current:</b> CRC calculation result as stored in CRC_DR register (32 bits).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DR LL_CRC_ReadData32</li> </ul>

### LL\_CRC\_ReadData16

Function Name	<code>__STATIC_INLINE uint16_t LL_CRC_ReadData16 (CRC_TypeDef * CRCx)</code>
---------------	--

Function Description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Current:</b> CRC calculation result as stored in CRC_DR register (16 bits).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function is expected to be used in a 16 bits CRC polynomial size context.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_CRC_ReadData16</li> </ul>

### LL\_CRC\_ReadData8

Function Name	<code>__STATIC_INLINE uint8_t LL_CRC_ReadData8 (CRC_TypeDef * CRCx)</code>
Function Description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Current:</b> CRC calculation result as stored in CRC_DR register (8 bits).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function is expected to be used in a 8 bits CRC polynomial size context.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_CRC_ReadData8</li> </ul>

### LL\_CRC\_ReadData7

Function Name	<code>__STATIC_INLINE uint8_t LL_CRC_ReadData7 (CRC_TypeDef * CRCx)</code>
Function Description	Return current CRC calculation result.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Current:</b> CRC calculation result as stored in CRC_DR register (7 bits).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function is expected to be used in a 7 bits CRC polynomial size context.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_CRC_ReadData7</li> </ul>

### LL\_CRC\_Read\_IDR

Function Name	<code>__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef * CRCx)</code>
Function Description	Return data stored in the Independent Data(IDR) register.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> stored in CRC_IDR register (General-purpose 8-bit</li> </ul>

---

	data register).
Notes	<ul style="list-style-type: none"> <li>This register can be used as a temporary storage location for one byte.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IDR IDR LL_CRC_Read_IDR</li> </ul>

### LL\_CRC\_Write\_IDR

Function Name	<code>_STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)</code>
Function Description	Store data in the Independent Data(IDR) register.
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> <li><b>InData:</b> value to be stored in CRC_IDR register (8-bit) between Min_Data=0 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This register can be used as a temporary storage location for one byte.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>IDR IDR LL_CRC_Write_IDR</li> </ul>

### LL\_CRC\_DeInit

Function Name	<code>ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)</code>
Function Description	De-initialize CRC registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none"> <li><b>CRCx:</b> CRC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: CRC registers are de-initialized</li> <li>ERROR: CRC registers are not de-initialized</li> </ul> </li> </ul>

## 57.2 CRC Firmware driver defines

### 57.2.1 CRC

#### *Default CRC computation initialization value*

`LL_CRC_DEFAULT_CRC_INITVALUE` Default CRC computation initialization value

#### *Default CRC generating polynomial value*

`LL_CRC_DEFAULT_CRC32_POLY` Default CRC generating polynomial value

#### *Input Data Reverse*

`LL_CRC_INDATA_REVERSE_NONE` Input Data bit order not affected

`LL_CRC_INDATA_REVERSE_BYTE` Input Data bit reversal done by byte

`LL_CRC_INDATA_REVERSE_HALFWORD` Input Data bit reversal done by half-word

**LL\_CRC\_INDATA\_REVERSE\_WORD** Input Data bit reversal done by word

***Output Data Reverse***

**LL\_CRC\_OUTDATA\_REVERSE\_NONE** Output Data bit order not affected

**LL\_CRC\_OUTDATA\_REVERSE\_BIT** Output Data bit reversal done by bit

***Polynomial length***

**LL\_CRC\_POLYLENGTH\_32B** 32 bits Polynomial size

**LL\_CRC\_POLYLENGTH\_16B** 16 bits Polynomial size

**LL\_CRC\_POLYLENGTH\_8B** 8 bits Polynomial size

**LL\_CRC\_POLYLENGTH\_7B** 7 bits Polynomial size

***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

## 58 LL CRS Generic Driver

### 58.1 CRS Firmware driver API description

#### 58.1.1 Detailed description of functions

##### **LL\_CRS\_EnableFreqErrorCounter**

Function Name	<code>__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter(void)</code>
Function Description	Enable Frequency error counter.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When this bit is set, the CRS_CFGR register is write-protected and cannot be modified</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR CEN LL_CRS_EnableFreqErrorCounter</li> </ul>

##### **LL\_CRS\_DisableFreqErrorCounter**

Function Name	<code>__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter(void)</code>
Function Description	Disable Frequency error counter.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR CEN LL_CRS_DisableFreqErrorCounter</li> </ul>

##### **LL\_CRS\_IsEnabledFreqErrorCounter**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter(void)</code>
Function Description	Check if Frequency error counter is enabled or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR CEN LL_CRS_IsEnabledFreqErrorCounter</li> </ul>

##### **LL\_CRS\_EnableAutoTrimming**

Function Name	<code>__STATIC_INLINE void LL_CRS_EnableAutoTrimming(void)</code>
Function Description	Enable Automatic trimming counter.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR AUTOTRIMEN LL_CRS_EnableAutoTrimming</li> </ul>

reference:

### **LL\_CRS\_DisableAutoTrimming**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void )</code></b>
Function Description	Disable Automatic trimming counter.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR AUTOTRIMEN LL_CRS_DisableAutoTrimming</li></ul>

### **LL\_CRS\_IsEnabledAutoTrimming**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void )</code></b>
Function Description	Check if Automatic trimming is enabled or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming</li></ul>

### **LL\_CRS\_SetHSI48SmoothTrimming**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)</code></b>
Function Description	Set HSI48 oscillator smooth trimming.
Parameters	<ul style="list-style-type: none"><li>• <b>Value:</b> a number between Min_Data = 0 and Max_Data = 63</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only</li><li>• Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR TRIM LL_CRS_SetHSI48SmoothTrimming</li></ul>

### **LL\_CRS\_GetHSI48SmoothTrimming**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void )</code></b>
Function Description	Get HSI48 oscillator smooth trimming.
Return values	<ul style="list-style-type: none"><li>• <b>a:</b> number between Min_Data = 0 and Max_Data = 63</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR TRIM LL_CRS_GetHSI48SmoothTrimming</li></ul>

**LL\_CRS\_SetReloadCounter**

Function Name	<code>__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)</code>
Function Description	Set counter reload value.
Parameters	<ul style="list-style-type: none"> <li>• <b>Value:</b> a number between Min_Data = 0 and Max_Data = 0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro <code>__LL_CRS_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)</code></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR RELOAD LL_CRS_SetReloadCounter</li> </ul>

**LL\_CRS\_GetReloadCounter**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void )</code>
Function Description	Get counter reload value.
Return values	<ul style="list-style-type: none"> <li>• <b>a:</b> number between Min_Data = 0 and Max_Data = 0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR RELOAD LL_CRS_GetReloadCounter</li> </ul>

**LL\_CRS\_SetFreqErrorLimit**

Function Name	<code>__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)</code>
Function Description	Set frequency error limit.
Parameters	<ul style="list-style-type: none"> <li>• <b>Value:</b> a number between Min_Data = 0 and Max_Data = 255</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR FELIM LL_CRS_SetFreqErrorLimit</li> </ul>

**LL\_CRS\_GetFreqErrorLimit**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void )</code>
Function Description	Get frequency error limit.
Return values	<ul style="list-style-type: none"> <li>• <b>A:</b> number between Min_Data = 0 and Max_Data = 255</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CFGR FELIM LL_CRS_GetFreqErrorLimit</li> </ul>

reference:

### **LL\_CRS\_SetSyncDivider**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)</code></b>
Function Description	Set division factor for SYNC signal.
Parameters	<ul style="list-style-type: none"> <li>• <b>Divider:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_CRS_SYNC_DIV_1</code></li> <li>- <code>LL_CRS_SYNC_DIV_2</code></li> <li>- <code>LL_CRS_SYNC_DIV_4</code></li> <li>- <code>LL_CRS_SYNC_DIV_8</code></li> <li>- <code>LL_CRS_SYNC_DIV_16</code></li> <li>- <code>LL_CRS_SYNC_DIV_32</code></li> <li>- <code>LL_CRS_SYNC_DIV_64</code></li> <li>- <code>LL_CRS_SYNC_DIV_128</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR SYNCDIV LL_CRS_SetSyncDivider</li> </ul>

### **LL\_CRS\_GetSyncDivider**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void )</code></b>
Function Description	Get division factor for SYNC signal.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_CRS_SYNC_DIV_1</code></li> <li>- <code>LL_CRS_SYNC_DIV_2</code></li> <li>- <code>LL_CRS_SYNC_DIV_4</code></li> <li>- <code>LL_CRS_SYNC_DIV_8</code></li> <li>- <code>LL_CRS_SYNC_DIV_16</code></li> <li>- <code>LL_CRS_SYNC_DIV_32</code></li> <li>- <code>LL_CRS_SYNC_DIV_64</code></li> <li>- <code>LL_CRS_SYNC_DIV_128</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR SYNCDIV LL_CRS_GetSyncDivider</li> </ul>

### **LL\_CRS\_SetSyncSignalSource**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)</code></b>
Function Description	Set SYNC signal source.
Parameters	<ul style="list-style-type: none"> <li>• <b>Source:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_CRS_SYNC_SOURCE_GPIO</code></li> <li>- <code>LL_CRS_SYNC_SOURCE_LSE</code></li> <li>- <code>LL_CRS_SYNC_SOURCE_USB</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to LL API cross reference:
- CFGR SYNC SRC LL\_CRS\_SetSyncSignalSource

### LL\_CRS\_GetSyncSignalSource

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource(void)</code>
Function Description	Get SYNC signal source.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_CRS_SYNC_SOURCE_GPIO</li> <li>- LL_CRS_SYNC_SOURCE_LSE</li> <li>- LL_CRS_SYNC_SOURCE_USB</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR SYNC SRC LL_CRS_SetSyncSignalSource</li> </ul>

### LL\_CRS\_SetSyncPolarity

Function Name	<code>__STATIC_INLINE void LL_CRS_SetSyncPolarity(uint32_t Polarity)</code>
Function Description	Set input polarity for the SYNC signal source.
Parameters	<ul style="list-style-type: none"> <li>• <b>Polarity:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_CRS_SYNC_POLARITY_RISING</li> <li>- LL_CRS_SYNC_POLARITY_FALLING</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR SYNC POL LL_CRS_SetSyncPolarity</li> </ul>

### LL\_CRS\_GetSyncPolarity

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity(void)</code>
Function Description	Get input polarity for the SYNC signal source.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_CRS_SYNC_POLARITY_RISING</li> <li>- LL_CRS_SYNC_POLARITY_FALLING</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR SYNC POL LL_CRS_SetSyncPolarity</li> </ul>

### LL\_CRS\_ConfigSynchronization

Function Name	<code>__STATIC_INLINE void LL_CRS_ConfigSynchronization(uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)</code>
Function Description	Configure CRS for the synchronization.

Parameters	<ul style="list-style-type: none"> <li><b>HSI48CalibrationValue:</b> a number between Min_Data = 0 and Max_Data = 63</li> <li><b>ErrorLimitValue:</b> a number between Min_Data = 0 and Max_Data = 0xFFFF</li> <li><b>ReloadValue:</b> a number between Min_Data = 0 and Max_Data = 255</li> <li><b>Settings:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128</li> <li>- LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB</li> <li>- LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR TRIM LL_CRS_ConfigSynchronization</li> <li>CFGR RELOAD LL_CRS_ConfigSynchronization</li> <li>CFGR FELIM LL_CRS_ConfigSynchronization</li> <li>CFGR SYNCDIV LL_CRS_ConfigSynchronization</li> <li>CFGR SYNCSRC LL_CRS_ConfigSynchronization</li> <li>CFGR SYNCPOL LL_CRS_ConfigSynchronization</li> </ul>

### LL\_CRS\_GenerateEvent\_SWSYNC

Function Name	<code>__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC( void )</code>
Function Description	Generate software SYNC event.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR SWSYNC LL_CRS_GenerateEvent_SWSYNC</li> </ul>

### LL\_CRS\_GetFreqErrorDirection

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection( void )</code>
Function Description	Get the frequency error direction latched in the time of the last SYNC event.
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_CRS_FREQ_ERROR_DIR_UP</li> <li>- LL_CRS_FREQ_ERROR_DIR_DOWN</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR FEDIR LL_CRS_GetFreqErrorDirection</li> </ul>

**LL\_CRS\_GetFreqErrorCapture**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void )</code>
Function Description	Get the frequency error counter value latched in the time of the last SYNC event.
Return values	<ul style="list-style-type: none"> <li>• <b>A:</b> number between Min_Data = 0x0000 and Max_Data = 0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR FECAP LL_CRS_GetFreqErrorCapture</li> </ul>

**LL\_CRS\_IsActiveFlag\_SYNCOK**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void )</code>
Function Description	Check if SYNC event OK signal occurred or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK</li> </ul>

**LL\_CRS\_IsActiveFlag\_SYNCWARN**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void )</code>
Function Description	Check if SYNC warning signal occurred or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN</li> </ul>

**LL\_CRS\_IsActiveFlag\_ERR**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void )</code>
Function Description	Check if Synchronization or trimming error signal occurred or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ERRF LL_CRS_IsActiveFlag_ERR</li> </ul>

**LL\_CRS\_IsActiveFlag\_ESYNC**

Function Name	<code>__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void )</code>
Function Description	Check if Expected SYNC signal occurred or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

Reference Manual to  
LL API cross  
reference:

- ISR ESYNCNF LL\_CRS\_IsActiveFlag\_ESYNC

### **LL\_CRS\_IsActiveFlag\_SYNCERR**

Function Name

**`_STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR  
(void )`**

Function Description

Check if SYNC error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- ISR SYNCERR LL\_CRS\_IsActiveFlag\_SYNCERR

### **LL\_CRS\_IsActiveFlag\_SYNCMISS**

Function Name

**`_STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS  
(void )`**

Function Description

Check if SYNC missed error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- ISR SYNCMISS LL\_CRS\_IsActiveFlag\_SYNCMISS

### **LL\_CRS\_IsActiveFlag\_TRIMOVF**

Function Name

**`_STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF  
(void )`**

Function Description

Check if Trimming overflow or underflow occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- ISR TRIMOVF LL\_CRS\_IsActiveFlag\_TRIMOVF

### **LL\_CRS\_ClearFlag\_SYNCOK**

Function Name

**`_STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void )`**

Function Description

Clear the SYNC event OK flag.

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- ICR SYNCOKC LL\_CRS\_ClearFlag\_SYNCOK

### **LL\_CRS\_ClearFlag\_SYNCWARN**

Function Name

**`_STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void  
)`**

---

Function Description	Clear the SYNC warning flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR SYNCWARN NC LL_CRS_ClearFlag_SYNCWARN</li> </ul>

### **LL\_CRS\_ClearFlag\_ERR**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void )</code></b>
Function Description	Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR ERRC LL_CRS_ClearFlag_ERR</li> </ul>

### **LL\_CRS\_ClearFlag\_ESYNC**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void )</code></b>
Function Description	Clear Expected SYNC flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR ESYNCC LL_CRS_ClearFlag_ESYNC</li> </ul>

### **LL\_CRS\_EnableIT\_SYNCOK**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void )</code></b>
Function Description	Enable SYNC event OK interrupt.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR SYNCOKIE LL_CRS_EnableIT_SYNCOK</li> </ul>

### **LL\_CRS\_DisableIT\_SYNCOK**

Function Name	<b><code>__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void )</code></b>
Function Description	Disable SYNC event OK interrupt.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR SYNCOKIE LL_CRS_DisableIT_SYNCOK</li> </ul>

### **LL\_CRS\_IsEnabledIT\_SYNCOK**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK</code></b>
---------------	--

**(void )**

Function Description	Check if SYNC event OK interrupt is enabled or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

**LL\_CRS\_EnableIT\_SYNCWARN**

Function Name	<b>__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void )</b>
Function Description	Enable SYNC warning interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

**LL\_CRS\_DisableIT\_SYNCWARN**

Function Name	<b>__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void )</b>
Function Description	Disable SYNC warning interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

**LL\_CRS\_IsEnabledIT\_SYNCWARN**

Function Name	<b>__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void )</b>
Function Description	Check if SYNC warning interrupt is enabled or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

**LL\_CRS\_EnableIT\_ERR**

Function Name	<b>__STATIC_INLINE void LL_CRS_EnableIT_ERR (void )</b>
Function Description	Enable Synchronization or trimming error interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	• CR ERRIE LL_CRS_EnableIT_ERR

**LL\_CRS\_DisableIT\_ERR**

Function Name **`__STATIC_INLINE void LL_CRS_DisableIT_ERR (void )`**

Function Description Disable Synchronization or trimming error interrupt.

Return values • **None:**

Reference Manual to  
LL API cross  
reference:  
• CR ERRIE LL\_CRS\_DisableIT\_ERR

**LL\_CRS\_IsEnabledIT\_ERR**

Function Name **`__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void )`**

Function Description Check if Synchronization or trimming error interrupt is enabled or not.

Return values • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
• CR ERRIE LL\_CRS\_IsEnabledIT\_ERR

**LL\_CRS\_EnableIT\_ESYNC**

Function Name **`__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void )`**

Function Description Enable Expected SYNC interrupt.

Return values • **None:**

Reference Manual to  
LL API cross  
reference:  
• CR ESYNCIE LL\_CRS\_EnableIT\_ESYNC

**LL\_CRS\_DisableIT\_ESYNC**

Function Name **`__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void )`**

Function Description Disable Expected SYNC interrupt.

Return values • **None:**

Reference Manual to  
LL API cross  
reference:  
• CR ESYNCIE LL\_CRS\_DisableIT\_ESYNC

**LL\_CRS\_IsEnabledIT\_ESYNC**

Function Name **`__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void )`**

Function Description Check if Expected SYNC interrupt is enabled or not.

Return values • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
• CR ESYNCIE LL\_CRS\_IsEnabledIT\_ESYNC

**LL\_CRS\_Delnit**

Function Name	<b>ErrorStatus LL_CRS_Delnit (void )</b>
Function Description	De-Initializes CRS peripheral registers to their default reset values.
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: CRS registers are de-initialized</li> <li>– ERROR: not applicable</li> </ul> </li> </ul>

**58.2 CRS Firmware driver defines****58.2.1 CRS*****Default Values***

LL\_CRS\_RELOADVALUE\_DEFAULT

**Notes:**

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL\_CRS\_ERRORLIMIT\_DEFAULT

LL\_CRS\_HSI48CALIBRATION\_DEFAULT

**Notes:**

- The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

***Frequency Error Direction***

LL\_CRS\_FREQ\_ERROR\_DIR\_UP

Upcounting direction, the actual frequency is above the target

LL\_CRS\_FREQ\_ERROR\_DIR\_DOWN

Downcounting direction, the actual frequency is below the target

***Get Flags Defines***

LL\_CRS\_ISR\_SYNCOKF

LL\_CRS\_ISR\_SYNCWARNF

LL\_CRS\_ISR\_ERRF

LL\_CRS\_ISR\_ESYNCF

LL\_CRS\_ISR\_SYNCERR

LL\_CRS\_ISR\_SYNCMISS

LL\_CRS\_ISR\_TRIMOVF

***IT Defines***

LL\_CRS\_CR\_SYNCOKIE

LL\_CRS\_CR\_SYNCWARNIE

---

`LL_CRS_CR_ERRIE`

`LL_CRS_CR_ESYNCIE`

**Synchronization Signal Divider**

`LL_CRS_SYNC_DIV_1`      Synchro Signal not divided (default)

`LL_CRS_SYNC_DIV_2`      Synchro Signal divided by 2

`LL_CRS_SYNC_DIV_4`      Synchro Signal divided by 4

`LL_CRS_SYNC_DIV_8`      Synchro Signal divided by 8

`LL_CRS_SYNC_DIV_16`      Synchro Signal divided by 16

`LL_CRS_SYNC_DIV_32`      Synchro Signal divided by 32

`LL_CRS_SYNC_DIV_64`      Synchro Signal divided by 64

`LL_CRS_SYNC_DIV_128`      Synchro Signal divided by 128

**Synchronization Signal Polarity**

`LL_CRS_SYNC_POLARITY_RISING`      Synchro Active on rising edge (default)

`LL_CRS_SYNC_POLARITY_FALLING`      Synchro Active on falling edge

**Synchronization Signal Source**

`LL_CRS_SYNC_SOURCE_GPIO`      Synchro Signal source GPIO

`LL_CRS_SYNC_SOURCE_LSE`      Synchro Signal source LSE

`LL_CRS_SYNC_SOURCE_USB`      Synchro Signal source USB SOF (default)

**Exported Macros\_Calculate\_Reload**

`_LL_CRS_CALCULATE_RELOADVALUE`      **Description:**

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

**Parameters:**

- `_FTARGET_`: Target frequency (value in Hz)
- `_FSYNC_`: Synchronization signal frequency (value in Hz)

**Return value:**

- Reload: value (in Hz)

**Notes:**

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET

**Common Write and read registers Macros****LL\_CRS\_WriteReg****Description:**

- Write a value in CRS register.

**Parameters:**

- \_\_INSTANCE\_\_: CRS Instance
- \_\_REG\_\_: Register to be written
- \_\_VALUE\_\_: Value to be written in the register

**Return value:**

- None

**LL\_CRS\_ReadReg****Description:**

- Read a value in CRS register.

**Parameters:**

- \_\_INSTANCE\_\_: CRS Instance
- \_\_REG\_\_: Register to be read

**Return value:**

- Register: value

## 59 LL DAC Generic Driver

### 59.1 DAC Firmware driver registers structures

#### 59.1.1 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()**.

### 59.2 DAC Firmware driver API description

#### 59.2.1 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	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>– LL_DAC_CHANNEL_1</li> <li>– LL_DAC_CHANNEL_2 (1)</li> </ul> </li> <li>• <b>TriggerSource:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DAC_TRIG_SOFTWARE</li> <li>– LL_DAC_TRIG_EXT_TIM2_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM3_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM3_CH3</li> <li>– LL_DAC_TRIG_EXT_TIM6_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM7_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM21_TRGO</li> <li>– LL_DAC_TRIG_EXT EXTI_LINE9</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().</li> <li>• To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.</li> <li>• Availability of parameters of trigger sources from timer depends on timers availability on the selected device.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR TSEL1 LL_DAC_SetTriggerSource</li> <li>• CR TSEL2 LL_DAC_SetTriggerSource</li> </ul>

### LL\_DAC\_GetTriggerSource

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource( (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Get the conversion trigger source for the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>– LL_DAC_CHANNEL_1</li> <li>– LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DAC_TRIG_SOFTWARE</li> <li>– LL_DAC_TRIG_EXT_TIM2_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM3_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM3_CH3</li> <li>– LL_DAC_TRIG_EXT_TIM6_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM7_TRGO</li> <li>– LL_DAC_TRIG_EXT_TIM21_TRGO</li> </ul> </li> </ul>

- 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	<code>__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration(DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)</code>
Function Description	Set the waveform automatic generation mode for the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> <li>• <b>WaveAutoGeneration:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DAC_WAVE_AUTO_GENERATION_NONE</li> <li>- LL_DAC_WAVE_AUTO_GENERATION_NOISE</li> <li>- LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WAVE1 LL_DAC_SetWaveAutoGeneration</li> <li>• CR WAVE2 LL_DAC_SetWaveAutoGeneration</li> </ul>

### LL\_DAC\_GetWaveAutoGeneration

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration(DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Get the waveform automatic generation mode for the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DAC_WAVE_AUTO_GENERATION_NONE</li> <li>- LL_DAC_WAVE_AUTO_GENERATION_NOISE</li> <li>- LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE</li> </ul> </li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CR WAVE1 LL_DAC_SetWaveAutoGeneration</li> </ul>

LL API cross  
reference:

- CR WAVE2 LL\_DAC\_SetWaveNoiseLFSR

### 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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL\_DAC\_CHANNEL\_1
  - LL\_DAC\_CHANNEL\_2 (1)
- **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: (1) On this STM32 serie, parameter not available on

Return values	<p>all devices. Refer to device datasheet for channels availability.</p> <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> <p><b>• Returned:</b> value can be one of the following values:</p> <ul style="list-style-type: none"> <li>- LL_DAC_NOISE_LFSR_UNMASK_BIT0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS1_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS2_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS3_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS4_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS5_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS6_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS7_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS8_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS9_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS10_0</li> <li>- LL_DAC_NOISE_LFSR_UNMASK_BITS11_0</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR MAMP1 LL_DAC_GetWaveNoiseLFSR</li> <li>• CR MAMP2 LL_DAC_GetWaveNoiseLFSR</li> </ul>

### LL\_DAC\_SetWaveTriangleAmplitude

Function Name	<code>__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude(DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)</code>
Function Description	Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> <li>• <b>TriangleAmplitude:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DAC_TRIANGLE_AMPLITUDE_1</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_3</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_7</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_15</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_31</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_63</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_127</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_255</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_511</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_1023</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_2047</li> <li>- LL_DAC_TRIANGLE_AMPLITUDE_4095</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Notes	<ul style="list-style-type: none"> <li>For wave generation to be effective, DAC channel wave generation mode must be enabled using function <code>LL_DAC_SetWaveAutoGeneration()</code>.</li> <li>This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR MAMP1 <code>LL_DAC_SetWaveTriangleAmplitude</code></li> <li>CR MAMP2 <code>LL_DAC_SetWaveTriangleAmplitude</code></li> </ul>

### LL\_DAC\_GetWaveTriangleAmplitude

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> <li><b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- <code>LL_DAC_CHANNEL_1</code></li> <li>- <code>LL_DAC_CHANNEL_2</code> (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_1</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_3</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_7</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_15</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_31</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_63</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_127</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_255</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_511</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_1023</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_2047</code></li> <li>- <code>LL_DAC_TRIANGLE_AMPLITUDE_4095</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR MAMP1 <code>LL_DAC_GetWaveTriangleAmplitude</code></li> <li>CR MAMP2 <code>LL_DAC_GetWaveTriangleAmplitude</code></li> </ul>

### LL\_DAC\_SetOutputBuffer

Function Name	<code>__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)</code>
Function Description	Set the output buffer for the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> <li><b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels</li> </ul>

	availability.
	<ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>OutputBuffer:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_DAC_OUTPUT_BUFFER_ENABLE</li> <li>- LL_DAC_OUTPUT_BUFFER_DISABLE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR BOFF1 LL_DAC_SetOutputBuffer</li> <li>• CR BOFF2 LL_DAC_SetOutputBuffer</li> </ul>

### LL\_DAC\_GetOutputBuffer

Function Name	<code>_STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer(DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Get the output buffer state for the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.           <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_DAC_OUTPUT_BUFFER_ENABLE</li> <li>- LL_DAC_OUTPUT_BUFFER_DISABLE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR BOFF1 LL_DAC_GetOutputBuffer</li> <li>• CR BOFF2 LL_DAC_GetOutputBuffer</li> </ul>

### LL\_DAC\_EnableDMAReq

Function Name	<code>_STATIC_INLINE void LL_DAC_EnableDMAReq(DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Enable DAC DMA transfer request of the selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.           <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR DMAEN1 LL_DAC_EnableDMAReq</li> <li>• CR DMAEN2 LL_DAC_EnableDMAReq</li> </ul>

reference:

### **LL\_DAC\_DisableDMAReq**

Function Name	<code>__STATIC_INLINE void LL_DAC_DisableDMAReq(DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Disable DAC DMA transfer request of the selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- <code>LL_DAC_CHANNEL_1</code></li> <li>- <code>LL_DAC_CHANNEL_2</code> (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To configure DMA source address (peripheral address), use function <code>LL_DAC_DMA_GetRegAddr()</code>.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR DMAEN1 <code>LL_DAC_DisableDMAReq</code></li> <li>• CR DMAEN2 <code>LL_DAC_DisableDMAReq</code></li> </ul>

### **LL\_DAC\_IsDMAReqEnabled**

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled(DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Get DAC DMA transfer request state of the selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- <code>LL_DAC_CHANNEL_1</code></li> <li>- <code>LL_DAC_CHANNEL_2</code> (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR DMAEN1 <code>LL_DAC_IsDMAReqEnabled</code></li> <li>• CR DMAEN2 <code>LL_DAC_IsDMAReqEnabled</code></li> </ul>

### **LL\_DAC\_DMA\_GetRegAddr**

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr(DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)</code>
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	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all</li> </ul>

	<p>devices. Refer to device datasheet for channels availability.</p> <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> <ul style="list-style-type: none"> <li>• <b>Register:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED</li> <li>- LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED</li> <li>- LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DAC:</b> register address</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example:  <code>LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1,    (uint32_t)&amp;&lt; array or variable &gt;,    LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1,    LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED),    LL_DMA_DIRECTION_MEMORY_TO_PERIPH);</code> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr</li> <li>• DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr</li> <li>• DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr</li> <li>• DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr</li> <li>• DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr</li> <li>• DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr</li> </ul>

### LL\_DAC\_Enable

Function Name	<code>__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Enable DAC selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.           <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR EN1 LL_DAC_Enable</li> <li>• CR EN2 LL_DAC_Enable</li> </ul>

### LL\_DAC\_Disable

Function Name	<code>__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
---------------	---

Function Description	Disable DAC selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>– LL_DAC_CHANNEL_1</li> <li>– LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR EN1 LL_DAC_Disable</li> <li>• CR EN2 LL_DAC_Disable</li> </ul>

### LL\_DAC\_IsEnabled

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_IsEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Get DAC enable state of the selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>– LL_DAC_CHANNEL_1</li> <li>– LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR EN1 LL_DAC_IsEnabled</li> <li>• CR EN2 LL_DAC_IsEnabled</li> </ul>

### LL\_DAC\_EnableTrigger

Function Name	<code>__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Enable DAC trigger of the selected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>– LL_DAC_CHANNEL_1</li> <li>– LL_DAC_CHANNEL_2 (1)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• - 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 or software trigger event is occurring. Select trigger</li> </ul>

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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL\_DAC\_CHANNEL\_1
  - LL\_DAC\_CHANNEL\_2 (1)

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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL\_DAC\_CHANNEL\_1
  - LL\_DAC\_CHANNEL\_2 (1)

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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.

	<ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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().</li> <li>• 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)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion</li> <li>• SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion</li> </ul>

### LL\_DAC\_ConvertData12RightAligned

Function Name	<code>_STATIC_INLINE void LL_DAC_ConvertData12RightAligned(DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)</code>
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	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul> </li> <li>• <b>Data:</b> Value between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned</li> <li>• DHR12R2 DACC2DHR</li> </ul>

### LL\_DAC\_ConvertData12LeftAligned

Function Name	<code>_STATIC_INLINE void LL_DAC_ConvertData12LeftAligned(DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)</code>
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	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> </ul> </li> </ul>

- LL\_DAC\_CHANNEL\_2 (1)
- **Data:** Value between Min\_Data=0x000 and Max\_Data=0xFFFF

Return values

Reference Manual to  
LL API cross  
reference:

- **None:**

- 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: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
  - LL\_DAC\_CHANNEL\_1
  - LL\_DAC\_CHANNEL\_2 (1)
- **Data:** Value between Min\_Data=0x00 and Max\_Data=0xFF

Return values

Reference Manual to  
LL API cross  
reference:

- **None:**

- 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=0xFFFF
- **DataChannel2:** Value between Min\_Data=0x000 and Max\_Data=0xFFFF

Return values

Reference Manual to  
LL API cross  
reference:

- **None:**

- DHR12RD DACC1DHR  
LL\_DAC\_ConvertDualData12RightAligned
- DHR12RD DACC2DHR  
LL\_DAC\_ConvertDualData12RightAligned

**LL\_DAC\_ConvertDualData12LeftAligned**

Function Name	<code>__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)</code>
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	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DataChannel1:</b> Value between Min_Data=0x000 and Max_Data=0xFFFF</li> <li>• <b>DataChannel2:</b> Value between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DHR12LD DACC1DHR <code>LL_DAC_ConvertDualData12LeftAligned</code></li> <li>• DHR12LD DACC2DHR <code>LL_DAC_ConvertDualData12LeftAligned</code></li> </ul>

**LL\_DAC\_ConvertDualData8RightAligned**

Function Name	<code>__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)</code>
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	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DataChannel1:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> <li>• <b>DataChannel2:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DHR8RD DACC1DHR <code>LL_DAC_ConvertDualData8RightAligned</code></li> <li>• DHR8RD DACC2DHR <code>LL_DAC_ConvertDualData8RightAligned</code></li> </ul>

**LL\_DAC\_RetrieveOutputData**

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)</code>
Function Description	Retrieve output data currently generated for the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> <li>• <b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability. <ul style="list-style-type: none"> <li>- <code>LL_DAC_CHANNEL_1</code></li> <li>- <code>LL_DAC_CHANNEL_2</code> (1)</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x000 and Max_Data=0xFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DOR1 DACC1DOR LL_DAC_RetrieveOutputData</li> <li>DOR2 DACC2DOR LL_DAC_RetrieveOutputData</li> </ul>

### LL\_DAC\_IsActiveFlag\_DMAUDR1

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1(DAC_TypeDef * DACx)</code>
Function Description	Get DAC underrun flag for DAC channel 1.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1</li> </ul>

### LL\_DAC\_IsActiveFlag\_DMAUDR2

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2(DAC_TypeDef * DACx)</code>
Function Description	Get DAC underrun flag for DAC channel 2.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2</li> </ul>

### LL\_DAC\_ClearFlag\_DMAUDR1

Function Name	<code>__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1(DAC_TypeDef * DACx)</code>
Function Description	Clear DAC underrun flag for DAC channel 1.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1</li> </ul>

### LL\_DAC\_ClearFlag\_DMAUDR2

Function Name	<code>__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2(DAC_TypeDef * DACx)</code>
---------------	--

Function Description	Clear DAC underrun flag for DAC channel 2.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

**LL\_DAC\_EnableIT\_DMAUDR1**

Function Name	<b><u>__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1(DAC_TypeDef * DACx)</u></b>
Function Description	Enable DMA underrun interrupt for DAC channel 1.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

**LL\_DAC\_EnableIT\_DMAUDR2**

Function Name	<b><u>__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2(DAC_TypeDef * DACx)</u></b>
Function Description	Enable DMA underrun interrupt for DAC channel 2.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

**LL\_DAC\_DisableIT\_DMAUDR1**

Function Name	<b><u>__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1(DAC_TypeDef * DACx)</u></b>
Function Description	Disable DMA underrun interrupt for DAC channel 1.
Parameters	<ul style="list-style-type: none"> <li>• <b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

**LL\_DAC\_DisableIT\_DMAUDR2**

Function Name	<b><u>__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2(DAC_TypeDef * DACx)</u></b>
Function Description	Disable DMA underrun interrupt for DAC channel 2.

---

Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	• CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

### LL\_DAC\_IsEnabledIT\_DMAUDR1

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1(DAC_TypeDef * DACx)</code>
Function Description	Get DMA underrun interrupt for DAC channel 1.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

### LL\_DAC\_IsEnabledIT\_DMAUDR2

Function Name	<code>__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2(DAC_TypeDef * DACx)</code>
Function Description	Get DMA underrun interrupt for DAC channel 2.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

### LL\_DAC\_DeInit

Function Name	<code>ErrorStatus LL_DAC_DeInit (DAC_TypeDef * DACx)</code>
Function Description	De-initialize registers of the selected DAC instance to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: DAC registers are de-initialized</li> <li>ERROR: not applicable</li> </ul> </li> </ul>

### LL\_DAC\_Init

Function Name	<code>ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, LL_DAC_InitTypeDef * DAC_InitStruct)</code>
Function Description	Initialize some features of DAC instance.
Parameters	<ul style="list-style-type: none"> <li><b>DACx:</b> DAC instance</li> <li><b>DAC_Channel:</b> This parameter can be one of the following values: (1) On this STM32 family, parameter not available on</li> </ul>

	all devices. Refer to device datasheet for channels availability.
	<ul style="list-style-type: none"> <li>- LL_DAC_CHANNEL_1</li> <li>- LL_DAC_CHANNEL_2 (1)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DAC_InitStruct:</b> Pointer to a LL_DAC_InitTypeDef structure</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>- SUCCESS: DAC registers are initialized</li> <li>- ERROR: DAC registers are not initialized</li> </ul> </li> <li>• The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC instance must be disabled.</li> </ul>

### LL\_DAC\_StructInit

Function Name	<b>void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)</b>
Function Description	Set each LL_DAC_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>DAC_InitStruct:</b> pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 59.3 DAC Firmware driver defines

### 59.3.1 DAC

#### *DAC channels*

LL\_DAC\_CHANNEL\_1    DAC channel 1

LL\_DAC\_CHANNEL\_2    DAC channel 2

#### *DAC flags*

LL\_DAC\_FLAG\_DMAUDR1    DAC channel 1 flag DMA underrun

LL\_DAC\_FLAG\_DMAUDR2    DAC channel 2 flag DMA underrun

#### *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 interruptions*

LL\_DAC\_IT\_DMAUDRIE1    DAC channel 1 interruption DMA underrun

LL\_DAC\_IT\_DMAUDRIE2    DAC channel 2 interruption DMA underrun

#### *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_TIM2_TRGO	DAC channel conversion trigger from external IP: TIM2 TRGO.
LL_DAC_TRIG_EXT_TIM3_TRGO	DAC channel conversion trigger from external IP: TIM3 TRGO.
LL_DAC_TRIG_EXT_TIM3_CH3	DAC channel conversion trigger from external IP: TIM3 CH3 event.
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_TIM21_TRGO	DAC channel conversion trigger from external IP: TIM21 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 (1)`

**Return value:**

- 1...2: (value "2" depending on DAC channel 2 availability)

**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 (value "2" depending on DAC channel 2 availability)

**Return value:**

- Returned: value can be one of the following values:
  - `LL_DAC_CHANNEL_1`
  - `LL_DAC_CHANNEL_2 (1)`

**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\\_DAT\\_A](#)**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

## 60 LL DMA Generic Driver

### 60.1 DMA Firmware driver registers structures

#### 60.1.1 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 PeriphRequest*
- *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::PeriphRequest**  
Specifies the peripheral request. This parameter can be a value of **DMA\_LL\_EC\_REQUEST**This feature can be modified afterwards using unitary function **LL\_DMA\_SetPeriphRequest()**.
  - **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()**.

## 60.2 DMA Firmware driver API description

### 60.2.1 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:

**LL\_DMA\_DisableChannel**

Function Name	<code>__STATIC_INLINE void LL_DMA_DisableChannel (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Disable DMA channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR EN LL_DMA_DisableChannel</li> </ul>

**LL\_DMA\_IsEnabledChannel**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Check if DMA channel is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR EN LL_DMA_IsEnabledChannel</li> </ul>

**LL\_DMA\_ConfigTransfer**

Function Name	<code>__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)</code>
Function Description	Configure all parameters link to DMA transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> </ul> </li> </ul>

- 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	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR CIRC LL_DMA_SetMode</li> </ul>

### LL\_DMA\_GetMode

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Get DMA mode circular or normal.
Parameters	<ul style="list-style-type: none"> <li><b>DMAx:</b> DMAx Instance</li> <li><b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_DMA_CHANNEL_1</li> <li>LL_DMA_CHANNEL_2</li> <li>LL_DMA_CHANNEL_3</li> <li>LL_DMA_CHANNEL_4</li> <li>LL_DMA_CHANNEL_5</li> <li>LL_DMA_CHANNEL_6</li> <li>LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>LL_DMA_MODE_NORMAL</li> <li>LL_DMA_MODE_CIRCULAR</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR CIRC LL_DMA_GetMode</li> </ul>

### LL\_DMA\_SetPeriphIncMode

Function Name	<code>__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)</code>
Function Description	Set Peripheral increment mode.
Parameters	<ul style="list-style-type: none"> <li><b>DMAx:</b> DMAx Instance</li> <li><b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_DMA_CHANNEL_1</li> <li>LL_DMA_CHANNEL_2</li> <li>LL_DMA_CHANNEL_3</li> <li>LL_DMA_CHANNEL_4</li> <li>LL_DMA_CHANNEL_5</li> <li>LL_DMA_CHANNEL_6</li> <li>LL_DMA_CHANNEL_7</li> </ul> </li> <li><b>PeriphOrM2MSrcIncMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_DMA_PERIPH_INCREMENT</li> <li>LL_DMA_PERIPH_NOINCREMENT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CCR PINC LL\_DMA\_SetPeriphIncMode

### **LL\_DMA\_GetPeriphIncMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode(DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Get Peripheral increment mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> <li>– LL_DMA_CHANNEL_3</li> <li>– LL_DMA_CHANNEL_4</li> <li>– LL_DMA_CHANNEL_5</li> <li>– LL_DMA_CHANNEL_6</li> <li>– LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_PERIPH_INCREMENT</li> <li>– LL_DMA_PERIPH_NOINCREMENT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PINC LL_DMA_GetPeriphIncMode</li> </ul>

### **LL\_DMA\_SetMemoryIncMode**

Function Name	<code>__STATIC_INLINE void LL_DMA_SetMemoryIncMode(DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)</code>
Function Description	Set Memory increment mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> <li>– LL_DMA_CHANNEL_3</li> <li>– LL_DMA_CHANNEL_4</li> <li>– LL_DMA_CHANNEL_5</li> <li>– LL_DMA_CHANNEL_6</li> <li>– LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>MemoryOrM2MDstIncMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_MEMORY_INCREMENT</li> <li>– LL_DMA_MEMORY_NOINCREMENT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR MINC LL_DMA_SetMemoryIncMode</li> </ul>

**LL\_DMA\_GetMemoryIncMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode(DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Get Memory increment mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> <li>– LL_DMA_CHANNEL_3</li> <li>– LL_DMA_CHANNEL_4</li> <li>– LL_DMA_CHANNEL_5</li> <li>– LL_DMA_CHANNEL_6</li> <li>– LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_MEMORY_INCREMENT</li> <li>– LL_DMA_MEMORY_NOINCREMENT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR MINC LL_DMA_GetMemoryIncMode</li> </ul>

**LL\_DMA\_SetPeriphSize**

Function Name	<code>__STATIC_INLINE void LL_DMA_SetPeriphSize(DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)</code>
Function Description	Set Peripheral size.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> <li>– LL_DMA_CHANNEL_3</li> <li>– LL_DMA_CHANNEL_4</li> <li>– LL_DMA_CHANNEL_5</li> <li>– LL_DMA_CHANNEL_6</li> <li>– LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>PeriphOrM2MSrcDataSize:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_PDATAALIGN_BYTE</li> <li>– LL_DMA_PDATAALIGN_HALFWORD</li> <li>– LL_DMA_PDATAALIGN_WORD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PSIZE LL_DMA_SetPeriphSize</li> </ul>

**LL\_DMA\_GetPeriphSize**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize</code>
---------------	--

**(DMA\_TypeDef \* DMAX, uint32\_t Channel)**

Function Description	Get Peripheral size.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAX:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_PDATAALIGN_BYTE</li> <li>- LL_DMA_PDATAALIGN_HALFWORD</li> <li>- LL_DMA_PDATAALIGN_WORD</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PSIZE LL_DMA_GetPeriphSize</li> </ul>

**LL\_DMA\_SetMemorySize**

Function Name	<b>_STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAX, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)</b>
Function Description	Set Memory size.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAX:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>MemoryOrM2MDstDataSize:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_MDATAALIGN_BYTE</li> <li>- LL_DMA_MDATAALIGN_HALFWORD</li> <li>- LL_DMA_MDATAALIGN_WORD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR MSIZE LL_DMA_SetMemorySize</li> </ul>

**LL\_DMA\_GetMemorySize**

Function Name	<b>_STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAX, uint32_t Channel)</b>
---------------	--

Function Description	Get Memory size.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_MDATAALIGN_BYTE</li> <li>- LL_DMA_MDATAALIGN_HALFWORD</li> <li>- LL_DMA_MDATAALIGN_WORD</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR MSIZE LL_DMA_GetMemorySize</li> </ul>

### LL\_DMA\_SetChannelPriorityLevel

Function Name	<b><code>_STATIC_INLINE void LL_DMA_SetChannelPriorityLevel(DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)</code></b>
Function Description	Set Channel priority level.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>Priority:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_PRIORITY_LOW</li> <li>- LL_DMA_PRIORITY_MEDIUM</li> <li>- LL_DMA_PRIORITY_HIGH</li> <li>- LL_DMA_PRIORITY_VERYHIGH</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PL LL_DMA_SetChannelPriorityLevel</li> </ul>

### LL\_DMA\_GetChannelPriorityLevel

Function Name	<b><code>_STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel(DMA_TypeDef * DMAx, uint32_t Channel)</code></b>
Function Description	Get Channel priority level.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:</li> </ul>

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_PRIORITY_LOW</li> <li>- LL_DMA_PRIORITY_MEDIUM</li> <li>- LL_DMA_PRIORITY_HIGH</li> <li>- LL_DMA_PRIORITY_VERYHIGH</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR PL LL_DMA_GetChannelPriorityLevel</li> </ul>

### LL\_DMA\_SetDataLength

Function Name	<code>__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMax, uint32_t Channel, uint32_t NbData)</code>
Function Description	Set Number of data to transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMax:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>NbData:</b> Between Min_Data = 0 and Max_Data = 0x0000FFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This action has no effect if channel is enabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CNDTR NDT LL_DMA_SetDataLength</li> </ul>

### LL\_DMA\_GetDataLength

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMax, uint32_t Channel)</code>
Function Description	Get Number of data to transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMax:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Between:</b> Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CNDTR NDT LL_DMA_GetDataLength</li> </ul>

### LL\_DMA\_ConfigAddresses

Function Name	<code>__STATIC_INLINE void LL_DMA_ConfigAddresses(DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)</code>
Function Description	Configure the Source and Destination addresses.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>SrcAddress:</b> Between Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> <li>• <b>DstAddress:</b> Between Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> <li>• <b>Direction:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_DIRECTION_PERIPH_TO_MEMORY</li> <li>- LL_DMA_DIRECTION_MEMORY_TO_PERIPH</li> <li>- LL_DMA_DIRECTION_MEMORY_TO_MEMORY</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Each IP using DMA provides an API to get directly the register address (LL_PPP_DMA_GetRegAddr)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CPAR PA LL_DMA_ConfigAddresses</li> <li>• CMAR MA LL_DMA_ConfigAddresses</li> </ul>

### LL\_DMA\_SetMemoryAddress

Function Name	<code>__STATIC_INLINE void LL_DMA_SetMemoryAddress(DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)</code>
Function Description	Set the Memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:</li> </ul>

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>MemoryAddress:</b> Between Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CMAR MA LL_DMA_SetMemoryAddress</li> </ul>

### LL\_DMA\_SetPeriphAddress

Function Name	<code>__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)</code>
Function Description	Set the Peripheral address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>PeriphAddress:</b> Between Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CPAR PA LL_DMA_SetPeriphAddress</li> </ul>

### LL\_DMA\_GetMemoryAddress

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Get Memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Between:</b> Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CMAR MA LL_DMA_GetMemoryAddress</li> </ul>

### LL\_DMA\_GetPeriphAddress

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Get Peripheral address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Between:</b> Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CPAR PA LL_DMA_GetPeriphAddress</li> </ul>

### LL\_DMA\_SetM2MSrcAddress

Function Name	<code>__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)</code>
Function Description	Set the Memory to Memory Source address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>MemoryAddress:</b> Between Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CPAR PA LL_DMA_SetM2MSrcAddress</li> </ul>

### LL\_DMA\_SetM2MDstAddress

Function Name	<code>_STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAX, uint32_t Channel, uint32_t MemoryAddress)</code>
Function Description	Set the Memory to Memory Destination address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAX:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>MemoryAddress:</b> Between Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CMAR MA LL_DMA_SetM2MDstAddress</li> </ul>

### LL\_DMA\_GetM2MSrcAddress

Function Name	<code>_STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAX, uint32_t Channel)</code>
Function Description	Get the Memory to Memory Source address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAX:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Between:</b> Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CPAR PA LL_DMA_GetM2MSrcAddress</li> </ul>

### LL\_DMA\_GetM2MDstAddress

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress(DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Get the Memory to Memory Destination address.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Between:</b> Min_Data = 0 and Max_Data = 0xFFFFFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CMAR MA LL_DMA_GetM2MDstAddress</li> </ul>

### LL\_DMA\_SetPeriphRequest

Function Name	<code>__STATIC_INLINE void LL_DMA_SetPeriphRequest(DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphRequest)</code>
Function Description	Set DMA request for DMA instance on Channel x.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> <li>• <b>PeriphRequest:</b> This parameter can be one of the following values:</li> </ul>

- LL\_DMA\_REQUEST\_0
- LL\_DMA\_REQUEST\_1
- LL\_DMA\_REQUEST\_2
- LL\_DMA\_REQUEST\_3
- LL\_DMA\_REQUEST\_4
- LL\_DMA\_REQUEST\_5
- LL\_DMA\_REQUEST\_6
- LL\_DMA\_REQUEST\_7
- LL\_DMA\_REQUEST\_8
- LL\_DMA\_REQUEST\_9
- LL\_DMA\_REQUEST\_10
- LL\_DMA\_REQUEST\_11
- LL\_DMA\_REQUEST\_12
- LL\_DMA\_REQUEST\_13
- LL\_DMA\_REQUEST\_14
- LL\_DMA\_REQUEST\_15

Return values

- **None:**

Notes

- Please refer to Reference Manual to get the available mapping of Request value link to Channel Selection.

Reference Manual to  
LL API cross  
reference:

- CSELR C1S LL\_DMA\_SetPeriphRequest
- CSELR C2S LL\_DMA\_SetPeriphRequest
- CSELR C3S LL\_DMA\_SetPeriphRequest
- CSELR C4S LL\_DMA\_SetPeriphRequest
- CSELR C5S LL\_DMA\_SetPeriphRequest
- CSELR C6S LL\_DMA\_SetPeriphRequest
- CSELR C7S LL\_DMA\_SetPeriphRequest

## LL\_DMA\_GetPeriphRequest

Function Name **`_STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest(DMA_TypeDef * DMax, uint32_t Channel)`**

Function Description Get DMA request for DMA instance on Channel x.

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\_REQUEST\_0
  - LL\_DMA\_REQUEST\_1
  - LL\_DMA\_REQUEST\_2
  - LL\_DMA\_REQUEST\_3
  - LL\_DMA\_REQUEST\_4
  - LL\_DMA\_REQUEST\_5
  - LL\_DMA\_REQUEST\_6
  - LL\_DMA\_REQUEST\_7

- LL\_DMA\_REQUEST\_8
- LL\_DMA\_REQUEST\_9
- LL\_DMA\_REQUEST\_10
- LL\_DMA\_REQUEST\_11
- LL\_DMA\_REQUEST\_12
- LL\_DMA\_REQUEST\_13
- LL\_DMA\_REQUEST\_14
- LL\_DMA\_REQUEST\_15

Reference Manual to  
LL API cross  
reference:

- CSELR C1S LL\_DMA\_GetPeriphRequest
- CSELR C2S LL\_DMA\_GetPeriphRequest
- CSELR C3S LL\_DMA\_GetPeriphRequest
- CSELR C4S LL\_DMA\_GetPeriphRequest
- CSELR C5S LL\_DMA\_GetPeriphRequest
- CSELR C6S LL\_DMA\_GetPeriphRequest
- CSELR C7S LL\_DMA\_GetPeriphRequest

### **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:

### LL\_DMA\_IsActiveFlag\_GI4

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 4 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li><b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>ISR GIF4 LL_DMA_IsActiveFlag_GI4</li></ul>

### LL\_DMA\_IsActiveFlag\_GI5

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 5 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li><b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>ISR GIF5 LL_DMA_IsActiveFlag_GI5</li></ul>

### LL\_DMA\_IsActiveFlag\_GI6

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 6 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li><b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>ISR GIF6 LL_DMA_IsActiveFlag_GI6</li></ul>

### LL\_DMA\_IsActiveFlag\_GI7

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 7 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li><b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>ISR GIF7 LL_DMA_IsActiveFlag_GI7</li></ul>

**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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 5 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TCIF5 LL_DMA_IsActiveFlag_TC5</li></ul>

**LL\_DMA\_IsActiveFlag\_TC6**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 6 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TCIF6 LL_DMA_IsActiveFlag_TC6</li></ul>

**LL\_DMA\_IsActiveFlag\_TC7**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 7 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TCIF7 LL_DMA_IsActiveFlag_TC7</li></ul>

**LL\_DMA\_IsActiveFlag\_HT1**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 1 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR HTIF1 LL_DMA_IsActiveFlag_HT1</li></ul>

**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  
reference:  
• ISR HTIF4 LL\_DMA\_IsActiveFlag\_HT4

**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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 6 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR HTIF6 LL_DMA_IsActiveFlag_HT6</li></ul>

**LL\_DMA\_IsActiveFlag\_HT7**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 7 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR HTIF7 LL_DMA_IsActiveFlag_HT7</li></ul>

**LL\_DMA\_IsActiveFlag\_TE1**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 1 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TEIF1 LL_DMA_IsActiveFlag_TE1</li></ul>

**LL\_DMA\_IsActiveFlag\_TE2**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 2 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TEIF2 LL_DMA_IsActiveFlag_TE2</li></ul>

**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	<code>__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7(DMA_TypeDef * DMAx)</code>
Function Description	Get Channel 7 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TEIF7 LL_DMA_IsActiveFlag_TE7</li></ul>

**LL\_DMA\_ClearFlag\_GI1**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI1(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 1 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CGIF1 LL_DMA_ClearFlag_GI1</li></ul>

**LL\_DMA\_ClearFlag\_GI2**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI2(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 2 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CGIF2 LL_DMA_ClearFlag_GI2</li></ul>

**LL\_DMA\_ClearFlag\_GI3**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI3(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 3 global interrupt flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CGIF3 LL_DMA_ClearFlag_GI3</li></ul>

**LL\_DMA\_ClearFlag\_GI4**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 4 global interrupt flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IFCR CGIF4 LL_DMA_ClearFlag_GI4</li> </ul>

**LL\_DMA\_ClearFlag\_GI5**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 5 global interrupt flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IFCR CGIF5 LL_DMA_ClearFlag_GI5</li> </ul>

**LL\_DMA\_ClearFlag\_GI6**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 6 global interrupt flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IFCR CGIF6 LL_DMA_ClearFlag_GI6</li> </ul>

**LL\_DMA\_ClearFlag\_GI7**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 7 global interrupt flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IFCR CGIF7 LL_DMA_ClearFlag_GI7</li> </ul>

**LL\_DMA\_ClearFlag\_TC1**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC1(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 1 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTCIF1 LL_DMA_ClearFlag_TC1</li></ul>

**LL\_DMA\_ClearFlag\_TC2**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC2(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 2 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTCIF2 LL_DMA_ClearFlag_TC2</li></ul>

**LL\_DMA\_ClearFlag\_TC3**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC3(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 3 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTCIF3 LL_DMA_ClearFlag_TC3</li></ul>

**LL\_DMA\_ClearFlag\_TC4**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TC4(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 4 transfer complete flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTCIF4 LL_DMA_ClearFlag_TC4</li></ul>

**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	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT2(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 2 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CHTIF2 LL_DMA_ClearFlag_HT2</li></ul>

**LL\_DMA\_ClearFlag\_HT3**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT3(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 3 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CHTIF3 LL_DMA_ClearFlag_HT3</li></ul>

**LL\_DMA\_ClearFlag\_HT4**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT4(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 4 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CHTIF4 LL_DMA_ClearFlag_HT4</li></ul>

**LL\_DMA\_ClearFlag\_HT5**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_HT5(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 5 half transfer flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CHTIF5 LL_DMA_ClearFlag_HT5</li></ul>

**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	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE3(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 3 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTEIF3 LL_DMA_ClearFlag_TE3</li></ul>

**LL\_DMA\_ClearFlag\_TE4**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE4(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 4 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTEIF4 LL_DMA_ClearFlag_TE4</li></ul>

**LL\_DMA\_ClearFlag\_TE5**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE5(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 5 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTEIF5 LL_DMA_ClearFlag_TE5</li></ul>

**LL\_DMA\_ClearFlag\_TE6**

Function Name	<code>__STATIC_INLINE void LL_DMA_ClearFlag_TE6(DMA_TypeDef * DMAx)</code>
Function Description	Clear Channel 6 transfer error flag.
Parameters	<ul style="list-style-type: none"><li>• <b>DMAx:</b> DMAx Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• IFCR CTEIF6 LL_DMA_ClearFlag_TE6</li></ul>

**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

reference:

### **LL\_DMA\_EnableIT\_TE**

Function Name	<code>__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Enable Transfer error interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR TEIE LL_DMA_EnableIT_TE</li> </ul>

### **LL\_DMA\_DisableIT\_TC**

Function Name	<code>__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Disable Transfer complete interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6</li> <li>- LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR TCIE LL_DMA_DisableIT_TC</li> </ul>

### **LL\_DMA\_DisableIT\_HT**

Function Name	<code>__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Disable Half transfer interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> </ul> </li> </ul>

- 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	<code>__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Check if Half transfer Interrupt is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> <li>– LL_DMA_CHANNEL_3</li> <li>– LL_DMA_CHANNEL_4</li> <li>– LL_DMA_CHANNEL_5</li> <li>– LL_DMA_CHANNEL_6</li> <li>– LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR HTIE LL_DMA_IsEnabledIT_HT</li> </ul>

**LL\_DMA\_IsEnabledIT\_TE**

Function Name	<code>__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	Check if Transfer error Interrupt is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> <li>– LL_DMA_CHANNEL_3</li> <li>– LL_DMA_CHANNEL_4</li> <li>– LL_DMA_CHANNEL_5</li> <li>– LL_DMA_CHANNEL_6</li> <li>– LL_DMA_CHANNEL_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR TEIE LL_DMA_IsEnabledIT_TE</li> </ul>

**LL\_DMA\_Init**

Function Name	<code>uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)</code>
Function Description	Initialize the DMA registers according to the specified parameters in DMA_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>– LL_DMA_CHANNEL_1</li> <li>– LL_DMA_CHANNEL_2</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6 (*)</li> <li>- LL_DMA_CHANNEL_7 (*)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DMA_InitStruct:</b> pointer to a LL_DMA_InitTypeDef structure.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: DMA registers are initialized</li> <li>- ERROR: Not applicable</li> </ul> </li> <li>• To convert DMAx_Channeln Instance to DMAx Instance and Channeln, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL</li> </ul>

### LL\_DMA\_DeInit

Function Name	<code>uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)</code>
Function Description	De-initialize the DMA registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMAx:</b> DMAx Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_DMA_CHANNEL_1</li> <li>- LL_DMA_CHANNEL_2</li> <li>- LL_DMA_CHANNEL_3</li> <li>- LL_DMA_CHANNEL_4</li> <li>- LL_DMA_CHANNEL_5</li> <li>- LL_DMA_CHANNEL_6 (*)</li> <li>- LL_DMA_CHANNEL_7 (*)</li> <li>- LL_DMA_CHANNEL_ALL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: DMA registers are de-initialized</li> <li>- ERROR: DMA registers are not de-initialized</li> </ul> </li> </ul>

### LL\_DMA\_StructInit

Function Name	<code>void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)</code>
Function Description	Set each LL_DMA_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>DMA_InitStruct:</b> Pointer to a LL_DMA_InitTypeDef structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 60.3 DMA Firmware driver defines

### 60.3.1 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
<b><i>Memory increment mode</i></b>	
LL_DMA_MEMORY_INCREMENT	Memory increment mode Enable
LL_DMA_MEMORY_NOINCREMENT	Memory increment mode Disable
<b><i>Transfer mode</i></b>	
LL_DMA_MODE_NORMAL	Normal Mode
LL_DMA_MODE_CIRCULAR	Circular Mode
<b><i>Peripheral data alignment</i></b>	
LL_DMA_PDATAALIGN_BYTE	Peripheral data alignment : Byte
LL_DMA_PDATAALIGN_HALFWORD	Peripheral data alignment : HalfWord
LL_DMA_PDATAALIGN_WORD	Peripheral data alignment : Word
<b><i>Peripheral increment mode</i></b>	
LL_DMA_PERIPH_INCREMENT	Peripheral increment mode Enable
LL_DMA_PERIPH_NOINCREMENT	Peripheral increment mode Disable
<b><i>Transfer Priority level</i></b>	
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
<b><i>Transfer peripheral request</i></b>	
LL_DMA_REQUEST_0	DMA peripheral request 0
LL_DMA_REQUEST_1	DMA peripheral request 1
LL_DMA_REQUEST_2	DMA peripheral request 2
LL_DMA_REQUEST_3	DMA peripheral request 3
LL_DMA_REQUEST_4	DMA peripheral request 4
LL_DMA_REQUEST_5	DMA peripheral request 5
LL_DMA_REQUEST_6	DMA peripheral request 6
LL_DMA_REQUEST_7	DMA peripheral request 7
LL_DMA_REQUEST_8	DMA peripheral request 8
LL_DMA_REQUEST_9	DMA peripheral request 9
LL_DMA_REQUEST_10	DMA peripheral request 10
LL_DMA_REQUEST_11	DMA peripheral request 11
LL_DMA_REQUEST_12	DMA peripheral request 12
LL_DMA_REQUEST_13	DMA peripheral request 13

`LL_DMA_REQUEST_14` DMA peripheral request 14

`LL_DMA_REQUEST_15` DMA peripheral request 15

### **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

## 61 LL EXTI Generic Driver

### 61.1 EXTI Firmware driver registers structures

#### 61.1.1 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\*\*](#).

## 61.2 EXTI Firmware driver API description

### 61.2.1 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\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_23
- LL\_EXTI\_LINE\_24
- LL\_EXTI\_LINE\_25
- LL\_EXTI\_LINE\_26
- LL\_EXTI\_LINE\_27
- LL\_EXTI\_LINE\_28
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31
- 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\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_23
- LL\_EXTI\_LINE\_24
- LL\_EXTI\_LINE\_25
- LL\_EXTI\_LINE\_26
- LL\_EXTI\_LINE\_27
- LL\_EXTI\_LINE\_28
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31
- 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

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\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_23
- LL\_EXTI\_LINE\_24
- LL\_EXTI\_LINE\_25
- LL\_EXTI\_LINE\_26
- LL\_EXTI\_LINE\_27
- LL\_EXTI\_LINE\_28
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31
- 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\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_23
- LL\_EXTI\_LINE\_24
- LL\_EXTI\_LINE\_25
- LL\_EXTI\_LINE\_26
- LL\_EXTI\_LINE\_27
- LL\_EXTI\_LINE\_28
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31
- 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\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_23
- LL\_EXTI\_LINE\_24
- LL\_EXTI\_LINE\_25
- LL\_EXTI\_LINE\_26
- LL\_EXTI\_LINE\_27
- LL\_EXTI\_LINE\_28
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31
- 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\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_23
- LL\_EXTI\_LINE\_24
- LL\_EXTI\_LINE\_25
- LL\_EXTI\_LINE\_26
- LL\_EXTI\_LINE\_27
- LL\_EXTI\_LINE\_28
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31
- 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
  - LL\_EXTI\_LINE\_20
  - LL\_EXTI\_LINE\_21
  - LL\_EXTI\_LINE\_22

	<ul style="list-style-type: none"> <li>- LL_EXTI_LINE_29</li> <li>- LL_EXTI_LINE_30</li> <li>- LL_EXTI_LINE_31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RTSR RTx LL_EXTI_EnableRisingTrig_0_31</li> </ul>

### LL\_EXTI\_DisableRisingTrig\_0\_31

Function Name	<code>__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)</code>
Function Description	Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.
Parameters	<ul style="list-style-type: none"> <li>• <b>ExtiLine:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_EXTI_LINE_0</li> <li>- LL_EXTI_LINE_1</li> <li>- LL_EXTI_LINE_2</li> <li>- LL_EXTI_LINE_3</li> <li>- LL_EXTI_LINE_4</li> <li>- LL_EXTI_LINE_5</li> <li>- LL_EXTI_LINE_6</li> <li>- LL_EXTI_LINE_7</li> <li>- LL_EXTI_LINE_8</li> <li>- LL_EXTI_LINE_9</li> <li>- LL_EXTI_LINE_10</li> <li>- LL_EXTI_LINE_11</li> <li>- LL_EXTI_LINE_12</li> <li>- LL_EXTI_LINE_13</li> <li>- LL_EXTI_LINE_14</li> <li>- LL_EXTI_LINE_15</li> <li>- LL_EXTI_LINE_16</li> <li>- LL_EXTI_LINE_18</li> <li>- LL_EXTI_LINE_19</li> <li>- LL_EXTI_LINE_20</li> <li>- LL_EXTI_LINE_21</li> <li>- LL_EXTI_LINE_22</li> <li>- LL_EXTI_LINE_29</li> <li>- LL_EXTI_LINE_30</li> <li>- LL_EXTI_LINE_31</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

---

Notes	<ul style="list-style-type: none"> <li>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.</li> <li>Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>RTSR RTx LL_EXTI_DisableRisingTrig_0_31</li> </ul>

### **LL\_EXTI\_IsEnabledRisingTrig\_0\_31**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)</code></b>
Function Description	Check if rising edge trigger is enabled for Lines in range 0 to 31.
Parameters	<ul style="list-style-type: none"> <li><b>ExtiLine:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- <code>LL_EXTI_LINE_0</code></li> <li>- <code>LL_EXTI_LINE_1</code></li> <li>- <code>LL_EXTI_LINE_2</code></li> <li>- <code>LL_EXTI_LINE_3</code></li> <li>- <code>LL_EXTI_LINE_4</code></li> <li>- <code>LL_EXTI_LINE_5</code></li> <li>- <code>LL_EXTI_LINE_6</code></li> <li>- <code>LL_EXTI_LINE_7</code></li> <li>- <code>LL_EXTI_LINE_8</code></li> <li>- <code>LL_EXTI_LINE_9</code></li> <li>- <code>LL_EXTI_LINE_10</code></li> <li>- <code>LL_EXTI_LINE_11</code></li> <li>- <code>LL_EXTI_LINE_12</code></li> <li>- <code>LL_EXTI_LINE_13</code></li> <li>- <code>LL_EXTI_LINE_14</code></li> <li>- <code>LL_EXTI_LINE_15</code></li> <li>- <code>LL_EXTI_LINE_16</code></li> <li>- <code>LL_EXTI_LINE_18</code></li> <li>- <code>LL_EXTI_LINE_19</code></li> <li>- <code>LL_EXTI_LINE_20</code></li> <li>- <code>LL_EXTI_LINE_21</code></li> <li>- <code>LL_EXTI_LINE_22</code></li> <li>- <code>LL_EXTI_LINE_29</code></li> <li>- <code>LL_EXTI_LINE_30</code></li> <li>- <code>LL_EXTI_LINE_31</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>RTSR RTx LL_EXTI_IsEnabledRisingTrig_0_31</li> </ul>

**LL\_EXTI\_EnableFallingTrig\_0\_31**

Function Name	<code>__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)</code>
Function Description	Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.
Parameters	<ul style="list-style-type: none"> <li>• <b>ExtiLine:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_EXTI_LINE_0</li> <li>- LL_EXTI_LINE_1</li> <li>- LL_EXTI_LINE_2</li> <li>- LL_EXTI_LINE_3</li> <li>- LL_EXTI_LINE_4</li> <li>- LL_EXTI_LINE_5</li> <li>- LL_EXTI_LINE_6</li> <li>- LL_EXTI_LINE_7</li> <li>- LL_EXTI_LINE_8</li> <li>- LL_EXTI_LINE_9</li> <li>- LL_EXTI_LINE_10</li> <li>- LL_EXTI_LINE_11</li> <li>- LL_EXTI_LINE_12</li> <li>- LL_EXTI_LINE_13</li> <li>- LL_EXTI_LINE_14</li> <li>- LL_EXTI_LINE_15</li> <li>- LL_EXTI_LINE_16</li> <li>- LL_EXTI_LINE_18</li> <li>- LL_EXTI_LINE_19</li> <li>- LL_EXTI_LINE_20</li> <li>- LL_EXTI_LINE_21</li> <li>- LL_EXTI_LINE_22</li> <li>- LL_EXTI_LINE_29</li> <li>- LL_EXTI_LINE_30</li> <li>- LL_EXTI_LINE_31</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• FTSR FTx LL_EXTI_EnableFallingTrig_0_31</li> </ul>

**LL\_EXTI\_DisableFallingTrig\_0\_31**

Function Name	<code>__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)</code>
Function Description	Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters	<ul style="list-style-type: none"> <li>• <b>ExtiLine:</b> This parameter can be a combination of the following values:           <ul style="list-style-type: none"> <li>- LL_EXTI_LINE_0</li> <li>- LL_EXTI_LINE_1</li> <li>- LL_EXTI_LINE_2</li> <li>- LL_EXTI_LINE_3</li> <li>- LL_EXTI_LINE_4</li> <li>- LL_EXTI_LINE_5</li> <li>- LL_EXTI_LINE_6</li> <li>- LL_EXTI_LINE_7</li> <li>- LL_EXTI_LINE_8</li> <li>- LL_EXTI_LINE_9</li> <li>- LL_EXTI_LINE_10</li> <li>- LL_EXTI_LINE_11</li> <li>- LL_EXTI_LINE_12</li> <li>- LL_EXTI_LINE_13</li> <li>- LL_EXTI_LINE_14</li> <li>- LL_EXTI_LINE_15</li> <li>- LL_EXTI_LINE_16</li> <li>- LL_EXTI_LINE_18</li> <li>- LL_EXTI_LINE_19</li> <li>- LL_EXTI_LINE_20</li> <li>- LL_EXTI_LINE_21</li> <li>- LL_EXTI_LINE_22</li> <li>- LL_EXTI_LINE_29</li> <li>- LL_EXTI_LINE_30</li> <li>- LL_EXTI_LINE_31</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• FTSR FTx LL_EXTI_DisableFallingTrig_0_31</li> </ul>

### LL\_EXTI\_IsEnabledFallingTrig\_0\_31

Function Name	<code>__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)</code>
Function Description	Check if falling edge trigger is enabled for Lines in range 0 to 31.
Parameters	<ul style="list-style-type: none"> <li>• <b>ExtiLine:</b> This parameter can be a combination of the following values:           <ul style="list-style-type: none"> <li>- LL_EXTI_LINE_0</li> <li>- LL_EXTI_LINE_1</li> <li>- LL_EXTI_LINE_2</li> <li>- LL_EXTI_LINE_3</li> </ul> </li> </ul>

- 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
- LL\_EXTI\_LINE\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_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:

- 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
- LL\_EXTI\_LINE\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22
- LL\_EXTI\_LINE\_29
- LL\_EXTI\_LINE\_30
- LL\_EXTI\_LINE\_31

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
- LL\_EXTI\_LINE\_20
- LL\_EXTI\_LINE\_21
- LL\_EXTI\_LINE\_22

	<ul style="list-style-type: none"> <li>- LL_EXTI_LINE_29</li> <li>- LL_EXTI_LINE_30</li> <li>- LL_EXTI_LINE_31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PR PIFx LL_EXTI_IsActiveFlag_0_31</li> </ul>

### LL\_EXTI\_ReadFlag\_0\_31

Function Name	<code>__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)</code>
Function Description	Read ExtLine Combination Flag for Lines in range 0 to 31.
Parameters	<ul style="list-style-type: none"> <li>• <b>ExtiLine:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_EXTI_LINE_0</li> <li>- LL_EXTI_LINE_1</li> <li>- LL_EXTI_LINE_2</li> <li>- LL_EXTI_LINE_3</li> <li>- LL_EXTI_LINE_4</li> <li>- LL_EXTI_LINE_5</li> <li>- LL_EXTI_LINE_6</li> <li>- LL_EXTI_LINE_7</li> <li>- LL_EXTI_LINE_8</li> <li>- LL_EXTI_LINE_9</li> <li>- LL_EXTI_LINE_10</li> <li>- LL_EXTI_LINE_11</li> <li>- LL_EXTI_LINE_12</li> <li>- LL_EXTI_LINE_13</li> <li>- LL_EXTI_LINE_14</li> <li>- LL_EXTI_LINE_15</li> <li>- LL_EXTI_LINE_16</li> <li>- LL_EXTI_LINE_18</li> <li>- LL_EXTI_LINE_19</li> <li>- LL_EXTI_LINE_20</li> <li>- LL_EXTI_LINE_21</li> <li>- LL_EXTI_LINE_22</li> <li>- LL_EXTI_LINE_29</li> <li>- LL_EXTI_LINE_30</li> <li>- LL_EXTI_LINE_31</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>@note:</b> This bit is set when the selected edge event arrives on the interrupt</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Please check each device line mapping for EXTI Line</li> </ul>

- availability
- Reference Manual to PR PIFx LL\_EXTI\_ReadFlag\_0\_31  
LL API cross reference:
- PR PIFx LL\_EXTI\_ReadFlag\_0\_31

### **LL\_EXTI\_ClearFlag\_0\_31**

Function Name	<code>__STATIC_INLINE void LL_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)</code>
Function Description	Clear ExtLine Flags for Lines in range 0 to 31.
Parameters	<ul style="list-style-type: none"> <li>• <b>ExtiLine:</b> This parameter can be a combination of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_EXTI_LINE_0</code></li> <li>- <code>LL_EXTI_LINE_1</code></li> <li>- <code>LL_EXTI_LINE_2</code></li> <li>- <code>LL_EXTI_LINE_3</code></li> <li>- <code>LL_EXTI_LINE_4</code></li> <li>- <code>LL_EXTI_LINE_5</code></li> <li>- <code>LL_EXTI_LINE_6</code></li> <li>- <code>LL_EXTI_LINE_7</code></li> <li>- <code>LL_EXTI_LINE_8</code></li> <li>- <code>LL_EXTI_LINE_9</code></li> <li>- <code>LL_EXTI_LINE_10</code></li> <li>- <code>LL_EXTI_LINE_11</code></li> <li>- <code>LL_EXTI_LINE_12</code></li> <li>- <code>LL_EXTI_LINE_13</code></li> <li>- <code>LL_EXTI_LINE_14</code></li> <li>- <code>LL_EXTI_LINE_15</code></li> <li>- <code>LL_EXTI_LINE_16</code></li> <li>- <code>LL_EXTI_LINE_18</code></li> <li>- <code>LL_EXTI_LINE_19</code></li> <li>- <code>LL_EXTI_LINE_20</code></li> <li>- <code>LL_EXTI_LINE_21</code></li> <li>- <code>LL_EXTI_LINE_22</code></li> <li>- <code>LL_EXTI_LINE_29</code></li> <li>- <code>LL_EXTI_LINE_30</code></li> <li>- <code>LL_EXTI_LINE_31</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Please check each device line mapping for EXTI Line availability</li> </ul>
Reference Manual to PR PIFx LL_EXTI_ClearFlag_0_31 LL API cross reference:	

### **LL\_EXTI\_Init**

Function Name	<code>uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)</code>
---------------	--

Function Description	Initialize the EXTI registers according to the specified parameters in <b>EXTI_InitStruct</b> .
Parameters	<ul style="list-style-type: none"> <li>• <b>EXTI_InitStruct:</b> pointer to a LL_EXTI_InitTypeDef structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: EXTI registers are initialized</li> <li>– ERROR: not applicable</li> </ul> </li> </ul>

**LL\_EXTI\_DeInit**

Function Name	<b>uint32_t LL_EXTI_DeInit (void )</b>
Function Description	De-initialize the EXTI registers to their default reset values.
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: EXTI registers are de-initialized</li> <li>– ERROR: not applicable</li> </ul> </li> </ul>

**LL\_EXTI\_StructInit**

Function Name	<b>void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)</b>
Function Description	Set each LL_EXTI_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>EXTI_InitStruct:</b> Pointer to a LL_EXTI_InitTypeDef structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 61.3 EXTI Firmware driver defines

### 61.3.1 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_19	Extended line 19
LL_EXTI_LINE_20	Extended line 20
LL_EXTI_LINE_21	Extended line 21
LL_EXTI_LINE_22	Extended line 22
LL_EXTI_LINE_23	Extended line 23
LL_EXTI_LINE_24	Extended line 24
LL_EXTI_LINE_25	Extended line 25
LL_EXTI_LINE_26	Extended line 26
LL_EXTI_LINE_28	Extended line 28
LL_EXTI_LINE_29	Extended line 29
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

## 62 LL GPIO Generic Driver

### 62.1 GPIO Firmware driver registers structures

#### 62.1.1 LL\_GPIO\_InitTypeDef

##### Data Fields

- *uint32\_t Pin*
- *uint32\_t Mode*
- *uint32\_t Speed*
- *uint32\_t OutputType*
- *uint32\_t Pull*
- *uint32\_t Alternate*

##### 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\(\)\*\*](#).
- ***uint32\_t LL\_GPIO\_InitTypeDef::Alternate***  
Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of [\*\*GPIO\\_LL\\_EC\\_AF\*\*](#).GPIO HW configuration can be modified afterwards using unitary function [\*\*LL\\_GPIO\\_SetAFPin\\_0\\_7\(\)\*\*](#) and [\*\*LL\\_GPIO\\_SetAFPin\\_8\\_15\(\)\*\*](#).

### 62.2 GPIO Firmware driver API description

#### 62.2.1 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	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>Pin:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> </ul> </li> <li>• <b>Mode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_MODE_INPUT</li> <li>- LL_GPIO_MODE_OUTPUT</li> <li>- LL_GPIO_MODE_ALTERNATE</li> <li>- LL_GPIO_MODE_ANALOG</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.</li> <li>• Warning: only one pin can be passed as parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• MODER MODEy LL_GPIO_SetPinMode</li> </ul>

**LL\_GPIO\_GetPinMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
Function Description	Return gpio mode for a dedicated pin on dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>Pin:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_GPIO_MODE_INPUT</li> <li>- LL_GPIO_MODE_OUTPUT</li> <li>- LL_GPIO_MODE_ALTERNATE</li> <li>- LL_GPIO_MODE_ANALOG</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.</li> <li>• Warning: only one pin can be passed as parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• MODER MODEy LL_GPIO_SetPinMode</li> </ul>

### LL\_GPIO\_SetPinOutputType

Function Name	<code>_STATIC_INLINE void LL_GPIO_SetPinOutputType(   GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t   OutputType)</code>
Function Description	Configure gpio output type for several pins on dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>PinMask:</b> This parameter can be a combination of the following values:           <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> <li>- LL_GPIO_PIN_ALL</li> </ul> </li> <li>• <b>OutputType:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_GPIO_OUTPUT_PUSHPULL</li> <li>- LL_GPIO_OUTPUT_OPENDRAIN</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Notes	<ul style="list-style-type: none"> <li>Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OTYPER OTy LL_GPIO_SetPinOutputType</li> </ul>

### LL\_GPIO\_SetPinOutputType

Function Name **\_STATIC\_INLINE uint32\_t LL\_GPIO\_SetPinOutputType  
(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:  
OTYPER OTy LL\_GPIO\_SetPinOutputType

### 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
  - LL\_GPIO\_SPEED\_FREQ\_VERY\_HIGH

Return values

- **None:**

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:

- OSPEEDR OSPEEDy LL\_GPIO\_SetPinSpeed

## 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	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_GPIO_SPEED_FREQ_LOW</li> <li>– LL_GPIO_SPEED_FREQ_MEDIUM</li> <li>– LL_GPIO_SPEED_FREQ_HIGH</li> <li>– LL_GPIO_SPEED_FREQ VERY_HIGH</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• I/O speed can be Low, Medium, Fast or High speed.</li> <li>• Warning: only one pin can be passed as parameter.</li> <li>• Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• OSPEEDR OSPEEDY LL_GPIO_GetPinSpeed</li> </ul>

### LL\_GPIO\_SetPinPull

Function Name	<b><code>__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)</code></b>
Function Description	Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>Pin:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_GPIO_PIN_0</li> <li>– LL_GPIO_PIN_1</li> <li>– LL_GPIO_PIN_2</li> <li>– LL_GPIO_PIN_3</li> <li>– LL_GPIO_PIN_4</li> <li>– LL_GPIO_PIN_5</li> <li>– LL_GPIO_PIN_6</li> <li>– LL_GPIO_PIN_7</li> <li>– LL_GPIO_PIN_8</li> <li>– LL_GPIO_PIN_9</li> <li>– LL_GPIO_PIN_10</li> <li>– LL_GPIO_PIN_11</li> <li>– LL_GPIO_PIN_12</li> <li>– LL_GPIO_PIN_13</li> <li>– LL_GPIO_PIN_14</li> <li>– LL_GPIO_PIN_15</li> </ul> </li> <li>• <b>Pull:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_GPIO_PULL_NO</li> <li>– LL_GPIO_PULL_UP</li> <li>– LL_GPIO_PULL_DOWN</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Warning: only one pin can be passed as parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PUPDR PUPDy LL_GPIO_SetPinPull</li> </ul>

**LL\_GPIO\_SetPinPull**

Function Name	<code>__STATIC_INLINE uint32_t LL_GPIO_SetPinPull( GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
Function Description	Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>Pin:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PULL_NO</li> <li>- LL_GPIO_PULL_UP</li> <li>- LL_GPIO_PULL_DOWN</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Warning: only one pin can be passed as parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PUPDR PUPDy LL_GPIO_SetPinPull</li> </ul>

**LL\_GPIO\_SetAFPin\_0\_7**

Function Name	<code>__STATIC_INLINE void LL_GPIO_SetAFPin_0_7( GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)</code>
Function Description	Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>Pin:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> </ul> </li> <li>• <b>Alternate:</b> This parameter can be one of the following</li> </ul>

values:

- LL\_GPIO\_AF\_0
- LL\_GPIO\_AF\_1
- LL\_GPIO\_AF\_2
- LL\_GPIO\_AF\_3
- LL\_GPIO\_AF\_4
- LL\_GPIO\_AF\_5
- LL\_GPIO\_AF\_6
- LL\_GPIO\_AF\_7

Return values

- **None:**

Notes

- Possible values are from AF0 to AF7 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to  
LL API cross  
reference:

- AFRL AFSELy LL\_GPIO\_SetAFPin\_0\_7

### **LL\_GPIO\_GetAFPin\_0\_7**

Function Name	<code>_STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
Function Description	Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>Pin:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_GPIO_AF_0</li> <li>- LL_GPIO_AF_1</li> <li>- LL_GPIO_AF_2</li> <li>- LL_GPIO_AF_3</li> <li>- LL_GPIO_AF_4</li> <li>- LL_GPIO_AF_5</li> <li>- LL_GPIO_AF_6</li> <li>- LL_GPIO_AF_7</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• AFRL AFSELy LL_GPIO_GetAFPin_0_7</li> </ul>

### **LL\_GPIO\_SetAFPin\_8\_15**

Function Name	<code>_STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)</code>
---------------	---

Function Description	Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> GPIO Port</li> <li><b>Pin:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> </ul> </li> <li><b>Alternate:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_AF_0</li> <li>- LL_GPIO_AF_1</li> <li>- LL_GPIO_AF_2</li> <li>- LL_GPIO_AF_3</li> <li>- LL_GPIO_AF_4</li> <li>- LL_GPIO_AF_5</li> <li>- LL_GPIO_AF_6</li> <li>- LL_GPIO_AF_7</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Possible values are from AF0 to AF7 depending on target.</li> <li>Warning: only one pin can be passed as parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>AFRH AFSELy LL_GPIO_SetAFPin_8_15</li> </ul>

### LL\_GPIO\_SetAFPin\_8\_15

Function Name	<code>_STATIC_INLINE uint32_t LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin)</code>
Function Description	Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> GPIO Port</li> <li><b>Pin:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_AF_0</li> <li>- LL_GPIO_AF_1</li> <li>- LL_GPIO_AF_2</li> <li>- LL_GPIO_AF_3</li> </ul> </li> </ul>

- LL\_GPIO\_AF\_4
- LL\_GPIO\_AF\_5
- LL\_GPIO\_AF\_6
- LL\_GPIO\_AF\_7

**Notes** • Possible values are from AF0 to AF7 depending on target.

**Reference Manual to LL API cross reference:** • AFRH AFSELy LL\_GPIO\_GetAFPin\_8\_15

## 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	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>PinMask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> <li>- LL_GPIO_PIN_ALL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• LCKR LCKY LL_GPIO_IsPinLocked</li> </ul>

### LL\_GPIO\_IsAnyPinLocked

Function Name	<code>_STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (GPIO_TypeDef * GPIOx)</code>
Function Description	Return 1 if one of the pin of a dedicated port is locked.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• LCKR LCKK LL_GPIO_IsAnyPinLocked</li> </ul>

### LL\_GPIO\_ReadInputPort

Function Name	<code>_STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (GPIO_TypeDef * GPIOx)</code>
Function Description	Return full input data register value for a dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Input:</b> data register value of port</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IDR IDy LL_GPIO_ReadInputPort</li> </ul>

**LL\_GPIO\_IsInputPinSet**

Function Name	<code>__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)</code>
Function Description	Return if input data level for several pins of dedicated port is high or low.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>PinMask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> <li>- LL_GPIO_PIN_ALL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IDR IDy LL_GPIO_IsInputPinSet</li> </ul>

**LL\_GPIO\_WriteOutputPort**

Function Name	<code>__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)</code>
Function Description	Write output data register for the port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>PortValue:</b> Level value for each pin of the port</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ODR ODy LL_GPIO_WriteOutputPort</li> </ul>

**LL\_GPIO\_ReadOutputPort**

Function Name	<code>__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)</code>
Function Description	Return full output data register value for a dedicated port.

---

Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> GPIO Port</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Output:</b> data register value of port</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ODR ODy LL_GPIO_ReadOutputPort</li> </ul>

### LL\_GPIO\_IsOutputPinSet

Function Name	<code>_STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)</code>
Function Description	Return if input data level for several pins of dedicated port is high or low.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> GPIO Port</li> <li><b>PinMask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> <li>- LL_GPIO_PIN_4</li> <li>- LL_GPIO_PIN_5</li> <li>- LL_GPIO_PIN_6</li> <li>- LL_GPIO_PIN_7</li> <li>- LL_GPIO_PIN_8</li> <li>- LL_GPIO_PIN_9</li> <li>- LL_GPIO_PIN_10</li> <li>- LL_GPIO_PIN_11</li> <li>- LL_GPIO_PIN_12</li> <li>- LL_GPIO_PIN_13</li> <li>- LL_GPIO_PIN_14</li> <li>- LL_GPIO_PIN_15</li> <li>- LL_GPIO_PIN_ALL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ODR ODy LL_GPIO_IsOutputPinSet</li> </ul>

### LL\_GPIO\_SetOutputPin

Function Name	<code>_STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)</code>
Function Description	Set several pins to high level on dedicated gpio port.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> GPIO Port</li> <li><b>PinMask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_GPIO_PIN_0</li> <li>- LL_GPIO_PIN_1</li> <li>- LL_GPIO_PIN_2</li> <li>- LL_GPIO_PIN_3</li> </ul> </li> </ul>

- 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	<code>_STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)</code>
Function Description	Toggle data value for several pin of dedicated port.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>PinMask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- <code>LL_GPIO_PIN_0</code></li> <li>- <code>LL_GPIO_PIN_1</code></li> <li>- <code>LL_GPIO_PIN_2</code></li> <li>- <code>LL_GPIO_PIN_3</code></li> <li>- <code>LL_GPIO_PIN_4</code></li> <li>- <code>LL_GPIO_PIN_5</code></li> <li>- <code>LL_GPIO_PIN_6</code></li> <li>- <code>LL_GPIO_PIN_7</code></li> <li>- <code>LL_GPIO_PIN_8</code></li> <li>- <code>LL_GPIO_PIN_9</code></li> <li>- <code>LL_GPIO_PIN_10</code></li> <li>- <code>LL_GPIO_PIN_11</code></li> <li>- <code>LL_GPIO_PIN_12</code></li> <li>- <code>LL_GPIO_PIN_13</code></li> <li>- <code>LL_GPIO_PIN_14</code></li> <li>- <code>LL_GPIO_PIN_15</code></li> <li>- <code>LL_GPIO_PIN_ALL</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ODR ODy LL_GPIO_TogglePin</li> </ul>

**LL\_GPIO\_DelInit**

Function Name	<code>ErrorStatus LL_GPIO_DelInit (GPIO_TypeDef * GPIOx)</code>
Function Description	De-initialize GPIO registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- <code>SUCCESS</code>: GPIO registers are de-initialized</li> <li>- <code>ERROR</code>: Wrong GPIO Port</li> </ul> </li> </ul>

**LL\_GPIO\_Init**

Function Name	<code>ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initialize GPIO registers according to the specified parameters in <code>GPIO_InitStruct</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> GPIO Port</li> <li>• <b>GPIO_InitStruct:</b> pointer to a <code>LL_GPIO_InitTypeDef</code> structure that contains the configuration information for the</li> </ul>

specified GPIO peripheral.

- |               |  |
|---------------|--|
| Return values | <ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content</li> <li>– ERROR: Not applicable</li> </ul> </li> </ul> |
|---------------|--|

### **LL\_GPIO\_StructInit**

Function Name	<b>void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)</b>
Function Description	Set each LL_GPIO_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_InitStruct:</b> pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## **62.3 GPIO Firmware driver defines**

### **62.3.1 GPIO**

#### ***Alternate Function***

<b>LL_GPIO_AF_0</b>	Select alternate function 0
<b>LL_GPIO_AF_1</b>	Select alternate function 1
<b>LL_GPIO_AF_2</b>	Select alternate function 2
<b>LL_GPIO_AF_3</b>	Select alternate function 3
<b>LL_GPIO_AF_4</b>	Select alternate function 4
<b>LL_GPIO_AF_5</b>	Select alternate function 5
<b>LL_GPIO_AF_6</b>	Select alternate function 6
<b>LL_GPIO_AF_7</b>	Select alternate function 7

#### ***Mode***

<b>LL_GPIO_MODE_INPUT</b>	Select input mode
<b>LL_GPIO_MODE_OUTPUT</b>	Select output mode
<b>LL_GPIO_MODE_ALTERNATE</b>	Select alternate function mode
<b>LL_GPIO_MODE_ANALOG</b>	Select analog mode

#### ***Output Type***

<b>LL_GPIO_OUTPUT_PUSHULL</b>	Select push-pull as output type
<b>LL_GPIO_OUTPUT_OPENDRAIN</b>	Select open-drain as output type

#### ***PIN***

<b>LL_GPIO_PIN_0</b>	Select pin 0
<b>LL_GPIO_PIN_1</b>	Select pin 1
<b>LL_GPIO_PIN_2</b>	Select pin 2
<b>LL_GPIO_PIN_3</b>	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_NO	Select I/O no pull
LL_GPIO_PULL_UP	Select I/O pull up
LL_GPIO_PULL_DOWN	Select I/O pull down

**Output Speed**

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 fast output speed
LL_GPIO_SPEED_FREQ VERY_HIGH	Select I/O high output speed

**Common Write and read registers Macros**

LL_GPIO_WriteReg	<b>Description:</b> • Write a value in GPIO register.
	<b>Parameters:</b> • __INSTANCE__: GPIO Instance • __REG__: Register to be written • __VALUE__: Value to be written in the register

**Return value:**

- None

LL_GPIO_ReadReg	<b>Description:</b>
	<b>Parameters:</b> • __INSTANCE__: GPIO Instance • __REG__: Register to be read

- Read a value in GPIO register.

**Parameters:**

- \_\_INSTANCE\_\_: GPIO Instance
- \_\_REG\_\_: Register to be read

**Return value:**

- Register: value

***GPIO Exported Constants***

LL\_GPIO\_SPEED\_LOW  
LL\_GPIO\_SPEED\_MEDIUM  
LL\_GPIO\_SPEED\_FAST  
LL\_GPIO\_SPEED\_HIGH

## 63 LL I2C Generic Driver

### 63.1 I2C Firmware driver registers structures

#### 63.1.1 LL\_I2C\_InitTypeDef

##### Data Fields

- *uint32\_t PeripheralMode*
- *uint32\_t Timing*
- *uint32\_t AnalogFilter*
- *uint32\_t DigitalFilter*
- *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::Timing***  
Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **\_LL\_I2C\_CONVERT\_TIMINGS()**This feature can be modified afterwards using unitary function **LL\_I2C\_SetTiming()**.
- ***uint32\_t LL\_I2C\_InitTypeDef::AnalogFilter***  
Enables or disables analog noise filter. This parameter can be a value of **I2C\_LL\_EC\_ANALOGFILTER\_SELECTION**This feature can be modified afterwards using unitary functions **LL\_I2C\_EnableAnalogFilter()** or **LL\_I2C\_DisableAnalogFilter()**.
- ***uint32\_t LL\_I2C\_InitTypeDef::DigitalFilter***  
Configures the digital noise filter. This parameter can be a number between Min\_Data = 0x00 and Max\_Data = 0x0FThis feature can be modified afterwards using unitary function **LL\_I2C\_SetDigitalFilter()**.
- ***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()**.

## 63.2 I2C Firmware driver API description

### 63.2.1 Detailed description of functions

#### LL\_I2C\_Enable

Function Name	<code>__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)</code>
Function Description	Enable I2C peripheral (PE = 1).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PE LL_I2C_Enable</li> </ul>

#### LL\_I2C\_Disable

Function Name	<code>__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)</code>
Function Description	Disable I2C peripheral (PE = 0).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PE LL_I2C_Disable</li> </ul>

#### LL\_I2C\_IsEnabled

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabled (I2C_TypeDef * I2Cx)</code>
Function Description	Check if the I2C peripheral is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PE LL_I2C_IsEnabled</li> </ul>

#### LL\_I2C\_ConfigFilters

Function Name	<code>__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)</code>
Function Description	Configure Noise Filters (Analog and Digital).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>AnalogFilter:</b> This parameter can be one of the following values:</li> </ul>

- LL\_I2C\_ANALOGFILTER\_ENABLE
  - LL\_I2C\_ANALOGFILTER\_DISABLE
  - **DigitalFilter:** This parameter must be a value between 0x00 (Digital filter disabled) and 0x0F (Digital filter enabled and filtering capability up to 15\*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]\*ti2cclk.
- |   |   |
|---|---|
| Return values                                     | • <b>None:</b>  |
| Notes   | • If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0). |
| Reference Manual to<br>LL API cross<br>reference: | <ul style="list-style-type: none"> <li>• CR1 ANFOFF LL_I2C_ConfigFilters</li> <li>• CR1 DNF LL_I2C_ConfigFilters</li> </ul>                                 |

### LL\_I2C\_SetDigitalFilter

Function Name	<b><code>__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)</code></b>
Function Description	Configure Digital Noise Filter.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>DigitalFilter:</b> This parameter must be a value between 0x00 (Digital filter disabled) and 0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.</li> </ul>
Return values	• <b>None:</b>
Notes	• If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).
Reference Manual to LL API cross reference:	• CR1 DNF LL_I2C_SetDigitalFilter

### LL\_I2C\_GetDigitalFilter

Function Name	<b><code>__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (I2C_TypeDef * I2Cx)</code></b>
Function Description	Get the current Digital Noise Filter configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x0 and Max_Data=0xF</li> </ul>
Reference Manual to LL API cross reference:	• CR1 DNF LL_I2C_GetDigitalFilter

**LL\_I2C\_EnableAnalogFilter**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_EnableAnalogFilter(I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable Analog Noise Filter.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This filter can only be programmed when the I2C is disabled (PE = 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ANFOFF LL_I2C_EnableAnalogFilter</li> </ul>

**LL\_I2C\_DisableAnalogFilter**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_DisableAnalogFilter(I2C_TypeDef * I2Cx)</code></b>
Function Description	Disable Analog Noise Filter.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This filter can only be programmed when the I2C is disabled (PE = 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ANFOFF LL_I2C_DisableAnalogFilter</li> </ul>

**LL\_I2C\_IsEnabledAnalogFilter**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter(I2C_TypeDef * I2Cx)</code></b>
Function Description	Check if Analog Noise Filter is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter</li> </ul>

**LL\_I2C\_EnableDMAReq\_TX**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_EnableDMAReq_TX(I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable DMA transmission requests.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CR1 TXDMAEN LL_I2C_EnableDMAReq_TX</li> </ul>

LL API cross  
reference:

### **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:  
CR1 TXDMAEN 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:  
CR1 TXDMAEN 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:  
CR1 RXDMAEN 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:  
CR1 RXDMAEN LL\_I2C\_DisableDMAReq\_RX

reference:

### **LL\_I2C\_IsEnabledDMAReq\_RX**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX(I2C_TypeDef * I2Cx)</code>
Function Description	Check if DMA reception requests are enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

### **LL\_I2C\_DMA\_GetRegAddr**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr(I2C_TypeDef * I2Cx, uint32_t Direction)</code>
Function Description	Get the data register address used for DMA transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance</li> <li>• <b>Direction:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_DMA_REG_DATA_TRANSMIT</li> <li>- LL_I2C_DMA_REG_DATA_RECEIVE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Address:</b> of data register</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TXDR TXDATA LL_I2C_DMA_GetRegAddr</li> <li>• RXDR RXDATA LL_I2C_DMA_GetRegAddr</li> </ul>

### **LL\_I2C\_EnableClockStretching**

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableClockStretching(I2C_TypeDef * I2Cx)</code>
Function Description	Enable Clock stretching.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit can only be programmed when the I2C is disabled (PE = 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 NOSTRETCH LL_I2C_EnableClockStretching</li> </ul>

### **LL\_I2C\_DisableClockStretching**

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableClockStretching(I2C_TypeDef * I2Cx)</code>
Function Description	Disable Clock stretching.

---

Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bit can only be programmed when the I2C is disabled (PE = 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 NOSTRETCH LL_I2C_DisableClockStretching</li> </ul>

### LL\_I2C\_IsEnabledClockStretching

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching(I2C_TypeDef * I2Cx)</code>
Function Description	Check if Clock stretching is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching</li> </ul>

### LL\_I2C\_EnableSlaveByteControl

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableSlaveByteControl(I2C_TypeDef * I2Cx)</code>
Function Description	Enable hardware byte control in slave mode.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 SBC LL_I2C_EnableSlaveByteControl</li> </ul>

### LL\_I2C\_DisableSlaveByteControl

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableSlaveByteControl(I2C_TypeDef * I2Cx)</code>
Function Description	Disable hardware byte control in slave mode.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 SBC LL_I2C_DisableSlaveByteControl</li> </ul>

### LL\_I2C\_IsEnabledSlaveByteControl

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl(I2C_TypeDef * I2Cx)</code>
---------------	--

Function Description	Check if hardware byte control in slave mode is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 SBC LL_I2C_IsEnabledSlaveByteControl

### LL\_I2C\_EnableWakeUpFromStop

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop(I2C_TypeDef * I2Cx)</code>
Function Description	Enable Wakeup from STOP.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.</li> <li>This bit can only be programmed when Digital Filter is disabled.</li> </ul>
Reference Manual to LL API cross reference:	CR1 WUPEN LL_I2C_EnableWakeUpFromStop

### LL\_I2C\_DisableWakeUpFromStop

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop(I2C_TypeDef * I2Cx)</code>
Function Description	Disable Wakeup from STOP.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	CR1 WUPEN LL_I2C_DisableWakeUpFromStop

### LL\_I2C\_IsEnabledWakeUpFromStop

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop(I2C_TypeDef * I2Cx)</code>
Function Description	Check if Wakeup from STOP is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>

---

Notes	<ul style="list-style-type: none"> <li>Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop</li> </ul>

### LL\_I2C\_EnableGeneralCall

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableGeneralCall(I2C_TypeDef * I2Cx)</code>
Function Description	Enable General Call.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When enabled the Address 0x00 is ACKed.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 GCEN LL_I2C_EnableGeneralCall</li> </ul>

### LL\_I2C\_DisableGeneralCall

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableGeneralCall(I2C_TypeDef * I2Cx)</code>
Function Description	Disable General Call.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When disabled the Address 0x00 is NACKed.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 GCEN LL_I2C_DisableGeneralCall</li> </ul>

### LL\_I2C\_IsEnabledGeneralCall

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall(I2C_TypeDef * I2Cx)</code>
Function Description	Check if General Call is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 GCEN LL_I2C_IsEnabledGeneralCall</li> </ul>

### LL\_I2C\_SetMasterAddressingMode

Function Name	<code>__STATIC_INLINE void LL_I2C_SetMasterAddressingMode(I2C_TypeDef * I2Cx, uint32_t AddressingMode)</code>
---------------	---

Function Description	Configure the Master to operate in 7-bit or 10-bit addressing mode.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>AddressingMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_I2C_ADDRESSING_MODE_7BIT</li> <li>– LL_I2C_ADDRESSING_MODE_10BIT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Changing this bit is not allowed, when the START bit is set.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 ADD10 LL_I2C_SetMasterAddressingMode</li> </ul>

### LL\_I2C\_SetMasterAddressingMode

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx)</code>
Function Description	Get the Master addressing mode.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_I2C_ADDRESSING_MODE_7BIT</li> <li>– LL_I2C_ADDRESSING_MODE_10BIT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 ADD10 LL_I2C_SetMasterAddressingMode</li> </ul>

### LL\_I2C\_SetOwnAddress1

Function Name	<code>__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)</code>
Function Description	Set the Own Address1.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>OwnAddress1:</b> This parameter must be a value between 0 and 0x3FF.</li> <li><b>OwnAddrSize:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_I2C_OWNADDRESS1_7BIT</li> <li>– LL_I2C_OWNADDRESS1_10BIT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR1 OA1 LL_I2C_SetOwnAddress1</li> <li>OAR1 OA1MODE LL_I2C_SetOwnAddress1</li> </ul>

### LL\_I2C\_EnableOwnAddress1

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)</code>
---------------	---

Function Description	Enable acknowledge on Own Address1 match address.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR1 OA1EN LL_I2C_EnableOwnAddress1</li> </ul>

### LL\_I2C\_DisableOwnAddress1

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)</code>
Function Description	Disable acknowledge on Own Address1 match address.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR1 OA1EN LL_I2C_DisableOwnAddress1</li> </ul>

### LL\_I2C\_IsEnabledOwnAddress1

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (I2C_TypeDef * I2Cx)</code>
Function Description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1</li> </ul>

### LL\_I2C\_SetOwnAddress2

Function Name	<code>__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)</code>
Function Description	Set the 7bits Own Address2.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>OwnAddress2:</b> Value between 0 and 0x7F.</li> <li><b>OwnAddrMask:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_I2C_OWNADDRESS2_NOMASK</li> <li>LL_I2C_OWNADDRESS2_MASK01</li> <li>LL_I2C_OWNADDRESS2_MASK02</li> <li>LL_I2C_OWNADDRESS2_MASK03</li> <li>LL_I2C_OWNADDRESS2_MASK04</li> <li>LL_I2C_OWNADDRESS2_MASK05</li> <li>LL_I2C_OWNADDRESS2_MASK06</li> <li>LL_I2C_OWNADDRESS2_MASK07</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This action has no effect if own address2 is enabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR2 OA2 LL_I2C_SetOwnAddress2</li> <li>OAR2 OA2MSK LL_I2C_SetOwnAddress2</li> </ul>

### LL\_I2C\_EnableOwnAddress2

Function Name	<b><code>__STATIC_INLINE void LL_I2C_EnableOwnAddress2(I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable acknowledge on Own Address2 match address.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR2 OA2EN LL_I2C_EnableOwnAddress2</li> </ul>

### LL\_I2C\_DisableOwnAddress2

Function Name	<b><code>__STATIC_INLINE void LL_I2C_DisableOwnAddress2(I2C_TypeDef * I2Cx)</code></b>
Function Description	Disable acknowledge on Own Address2 match address.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR2 OA2EN LL_I2C_DisableOwnAddress2</li> </ul>

### LL\_I2C\_IsEnabledOwnAddress2

Function Name	<b><code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2(I2C_TypeDef * I2Cx)</code></b>
Function Description	Check if Own Address1 acknowledge is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2</li> </ul>

### LL\_I2C\_SetTiming

Function Name	<b><code>__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)</code></b>
Function Description	Configure the SDA setup, hold time and the SCL high, low period.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>Timing:</b> This parameter must be a value between 0 and</li> </ul>

0xFFFFFFFF.

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bit can only be programmed when the I2C is disabled (PE = 0).</li> <li>This parameter is computed with the STM32CubeMX Tool.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMINGR TIMINGR LL_I2C_SetTiming</li> </ul>

### LL\_I2C\_GetTimingPrescaler

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler(I2C_TypeDef * I2Cx)</code>
Function Description	Get the Timing Prescaler setting.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x0 and Max_Data=0xF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMINGR PRESC LL_I2C_GetTimingPrescaler</li> </ul>

### LL\_I2C\_GetClockLowPeriod

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod(I2C_TypeDef * I2Cx)</code>
Function Description	Get the SCL low period setting.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMINGR SCLL LL_I2C_GetClockLowPeriod</li> </ul>

### LL\_I2C\_GetClockHighPeriod

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod(I2C_TypeDef * I2Cx)</code>
Function Description	Get the SCL high period setting.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMINGR SCLH LL_I2C_GetClockHighPeriod</li> </ul>

### LL\_I2C\_GetDataHoldTime

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime(I2C_TypeDef * I2Cx)</code>
---------------	--

Function Description	Get the SDA hold time.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x0 and Max_Data=0xF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMINGR SDADEL LL_I2C_GetDataHoldTime</li> </ul>

### LL\_I2C\_GetDataSetupTime

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime(I2C_TypeDef * I2Cx)</code>
Function Description	Get the SDA setup time.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x0 and Max_Data=0xF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMINGR SCLDEL LL_I2C_GetDataSetupTime</li> </ul>

### LL\_I2C\_SetMode

Function Name	<code>__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)</code>
Function Description	Configure peripheral mode.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>PeripheralMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_MODE_I2C</li> <li>- LL_I2C_MODE_SMBUS_HOST</li> <li>- LL_I2C_MODE_SMBUS_DEVICE</li> <li>- LL_I2C_MODE_SMBUS_DEVICE_ARP</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 SMBHEN LL_I2C_SetMode</li> <li>CR1 SMBDEN LL_I2C_SetMode</li> </ul>

### LL\_I2C\_GetMode

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)</code>
Function Description	Get peripheral mode.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_MODE_I2C</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_I2C_MODE_SMBUS_HOST</li> <li>- LL_I2C_MODE_SMBUS_DEVICE</li> <li>- LL_I2C_MODE_SMBUS_DEVICE_ARP</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 SMBHEN LL_I2C_GetMode</li> <li>• CR1 SMBDEN LL_I2C_GetMode</li> </ul>

### LL\_I2C\_EnableSMBusAlert

Function Name	<b><code>_STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable SMBus alert (Host or Device mode)
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>• SMBus Device mode: SMBus Alert pin is driven low and Alert Response Address Header acknowledge is enabled. SMBus Host mode: SMBus Alert pin management is supported.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ALERTEN LL_I2C_EnableSMBusAlert</li> </ul>

### LL\_I2C\_DisableSMBusAlert

Function Name	<b><code>_STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)</code></b>
Function Description	Disable SMBus alert (Host or Device mode)
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>• SMBus Device mode: SMBus Alert pin is not driven (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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ALERTEN LL_I2C_DisableSMBusAlert</li> </ul>

### LL\_I2C\_IsEnabledSMBusAlert

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert</code></b>
---------------	--

**(I2C\_TypeDef \* I2Cx)**

Function Description	Check if SMBus alert (Host or Device mode) is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert</li> </ul>

**LL\_I2C\_EnableSMBusPEC**

Function Name	<b>_STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)</b>
Function Description	Enable SMBus Packet Error Calculation (PEC).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PECEN LL_I2C_EnableSMBusPEC</li> </ul>

**LL\_I2C\_DisableSMBusPEC**

Function Name	<b>_STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)</b>
Function Description	Disable SMBus Packet Error Calculation (PEC).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PECEN LL_I2C_DisableSMBusPEC</li> </ul>

**LL\_I2C\_IsEnabledSMBusPEC**

Function Name	<b>_STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (I2C_TypeDef * I2Cx)</b>
Function Description	Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 PECEN LL_I2C_IsEnabledSMBusPEC</li> </ul>

### LL\_I2C\_ConfigSMBusTimeout

Function Name	<code>__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout(I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)</code>
Function Description	Configure the SMBus Clock Timeout.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>TimeoutA:</b> This parameter must be a value between Min_Data=0 and Max_Data=0xFFFF.</li> <li><b>TimeoutAMode:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW</li> <li>- LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH</li> </ul> </li> <li><b>TimeoutB:</b></li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/orTimeoutB).</li> <li>This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/orTimeoutB).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout</li> <li>TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout</li> <li>TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout</li> </ul>

### LL\_I2C\_SetSMBusTimeoutA

Function Name	<code>__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA(I2C_TypeDef * I2Cx, uint32_t TimeoutA)</code>
Function Description	Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>TimeoutA:</b> This parameter must be a value between Min_Data=0 and Max_Data=0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>

- These bits can only be programmed when TimeoutA is disabled.
  - TIMEOUTTR TIMEOUTA LL\_I2C\_SetSMBusTimeoutA
- Reference Manual to LL API cross reference:

### **LL\_I2C\_GetSMBusTimeoutA**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA(I2C_TypeDef * I2Cx)</code></b>
Function Description	Get the SMBus Clock TimeoutA setting.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0 and Max_Data=0FFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TIMEOUTTR TIMEOUTA LL_I2C_GetSMBusTimeoutA</li> </ul>

### **LL\_I2C\_SetSMBusTimeoutAMode**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode(I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)</code></b>
Function Description	Set the SMBus Clock TimeoutA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>TimeoutAMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW</li> <li>- LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>• This bit can only be programmed when TimeoutA is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TIMEOUTTR TIDLE LL_I2C_SetSMBusTimeoutAMode</li> </ul>

### **LL\_I2C\_GetSMBusTimeoutAMode**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode(I2C_TypeDef * I2Cx)</code></b>
Function Description	Get the SMBus Clock TimeoutA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW</li> <li>– LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode</li> </ul>

### LL\_I2C\_SetSMBusTimeoutB

Function Name	<code>__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB(I2C_TypeDef * I2Cx, uint32_t TimeoutB)</code>
Function Description	Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>TimeoutB:</b> This parameter must be a value between Min_Data=0 and Max_Data=0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>These bits can only be programmed when TimeoutB is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB</li> </ul>

### LL\_I2C\_GetSMBusTimeoutB

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB(I2C_TypeDef * I2Cx)</code>
Function Description	Get the SMBus Extended Cumulative Clock TimeoutB setting.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0 and Max_Data=0xFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB</li> </ul>

### LL\_I2C\_EnableSMBusTimeout

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableSMBusTimeout(I2C_TypeDef * I2Cx, uint32_t ClockTimeout)</code>
Function Description	Enable the SMBus Clock Timeout.

Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>ClockTimeout:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_SMBUS_TIMEOUTA</li> <li>– LL_I2C_SMBUS_TIMEOUTB</li> <li>– LL_I2C_SMBUS_ALL_TIMEOUT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout</li> <li>TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout</li> </ul>

### LL\_I2C\_DisableSMBusTimeout

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableSMBusTimeout(I2C_TypeDef * I2Cx, uint32_t ClockTimeout)</code>
Function Description	Disable the SMBus Clock Timeout.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>ClockTimeout:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_SMBUS_TIMEOUTA</li> <li>– LL_I2C_SMBUS_TIMEOUTB</li> <li>– LL_I2C_SMBUS_ALL_TIMEOUT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout</li> <li>TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout</li> </ul>

### LL\_I2C\_IsEnabledSMBusTimeout

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout(I2C_TypeDef * I2Cx, uint32_t ClockTimeout)</code>
Function Description	Check if the SMBus Clock Timeout is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>ClockTimeout:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_SMBUS_TIMEOUTA</li> <li>– LL_I2C_SMBUS_TIMEOUTB</li> <li>– LL_I2C_SMBUS_ALL_TIMEOUT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx</li> </ul>

Instance.

Reference Manual to  
LL API cross  
reference:

- TIMEOUTR TIMOUTEN LL\_I2C\_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL\_I2C\_IsEnabledSMBusTimeout

### **LL\_I2C\_EnableIT\_TX**

Function Name      **`_STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)`**

Function Description      Enable TXIS interrupt.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:

- CR1 TXIE LL\_I2C\_EnableIT\_TX

### **LL\_I2C\_DisableIT\_TX**

Function Name      **`_STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)`**

Function Description      Disable TXIS interrupt.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:

- CR1 TXIE 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 TXIS Interrupt is enabled or disabled.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- CR1 TXIE 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:
- CR1 RXIE LL\_I2C\_EnableIT\_RX

### **LL\_I2C\_DisableIT\_RX**

Function Name	<code>_STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)</code>
Function Description	Disable RXNE interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to  
LL API cross  
reference:

- CR1 RXIE LL\_I2C\_DisableIT\_RX

### **LL\_I2C\_IsEnabledIT\_RX**

Function Name	<code>_STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (I2C_TypeDef * I2Cx)</code>
Function Description	Check if the RXNE Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

Reference Manual to  
LL API cross  
reference:

- CR1 RXIE LL\_I2C\_IsEnabledIT\_RX

### **LL\_I2C\_EnableIT\_ADDR**

Function Name	<code>_STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)</code>
Function Description	Enable Address match interrupt (slave mode only).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to  
LL API cross  
reference:

- CR1 ADDRIE LL\_I2C\_EnableIT\_ADDR

### **LL\_I2C\_DisableIT\_ADDR**

Function Name	<code>_STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)</code>
Function Description	Disable Address match interrupt (slave mode only).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to  
LL API cross

- CR1 ADDRIE LL\_I2C\_DisableIT\_ADDR

reference:

### **LL\_I2C\_IsEnabledIT\_ADDR**

Function Name      **\_\_STATIC\_INLINE uint32\_t LL\_I2C\_IsEnabledIT\_ADDR (I2C\_TypeDef \* I2Cx)**

Function Description      Check if Address match interrupt is enabled or disabled.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
CR1 ADDRIE LL\_I2C\_IsEnabledIT\_ADDR

### **LL\_I2C\_EnableIT\_NACK**

Function Name      **\_\_STATIC\_INLINE void LL\_I2C\_EnableIT\_NACK (I2C\_TypeDef \* I2Cx)**

Function Description      Enable Not acknowledge received interrupt.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
CR1 NACKIE LL\_I2C\_EnableIT\_NACK

### **LL\_I2C\_DisableIT\_NACK**

Function Name      **\_\_STATIC\_INLINE void LL\_I2C\_DisableIT\_NACK (I2C\_TypeDef \* I2Cx)**

Function Description      Disable Not acknowledge received interrupt.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
CR1 NACKIE LL\_I2C\_DisableIT\_NACK

### **LL\_I2C\_IsEnabledIT\_NACK**

Function Name      **\_\_STATIC\_INLINE uint32\_t LL\_I2C\_IsEnabledIT\_NACK (I2C\_TypeDef \* I2Cx)**

Function Description      Check if Not acknowledge received interrupt is enabled or disabled.

Parameters      • **I2Cx:** I2C Instance.

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
CR1 NACKIE LL\_I2C\_IsEnabledIT\_NACK

reference:

### **LL\_I2C\_EnableIT\_STOP**

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)</code>
Function Description	Enable STOP detection interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 STOPIE LL_I2C_EnableIT_STOP</li> </ul>

### **LL\_I2C\_DisableIT\_STOP**

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)</code>
Function Description	Disable STOP detection interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 STOPIE LL_I2C_DisableIT_STOP</li> </ul>

### **LL\_I2C\_IsEnabledIT\_STOP**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (I2C_TypeDef * I2Cx)</code>
Function Description	Check if STOP detection interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 STOPIE LL_I2C_IsEnabledIT_STOP</li> </ul>

### **LL\_I2C\_EnableIT\_TC**

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)</code>
Function Description	Enable Transfer Complete interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CR1 TCIE LL_I2C_EnableIT_TC</li> </ul>

LL API cross  
reference:

### **LL\_I2C\_DisableIT\_TC**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)</code></b>
Function Description	Disable Transfer Complete interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TCIE LL_I2C_DisableIT_TC</li> </ul>

### **LL\_I2C\_IsEnabledIT\_TC**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (I2C_TypeDef * I2Cx)</code></b>
Function Description	Check if Transfer Complete interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TCIE LL_I2C_IsEnabledIT_TC</li> </ul>

### **LL\_I2C\_EnableIT\_ERR**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable Error interrupts.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)</li> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ERRIE LL_I2C_EnableIT_ERR</li> </ul>

**LL\_I2C\_DisableIT\_ERR**

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)</code>
Function Description	Disable Error interrupts.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)</li> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ERRIE LL_I2C_DisableIT_ERR</li> </ul>

**LL\_I2C\_IsEnabledIT\_ERR**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (I2C_TypeDef * I2Cx)</code>
Function Description	Check if Error interrupts is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ERRIE LL_I2C_IsEnabledIT_ERR</li> </ul>

**LL\_I2C\_IsActiveFlag\_TXE**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Transmit data register empty flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: When next data is written in Transmit data register.</li> <li>• SET: When Transmit data register is empty.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TXE LL_I2C_IsActiveFlag_TXE</li> </ul>

**LL\_I2C\_IsActiveFlag\_TXIS**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (I2C_TypeDef * I2Cx)</code>
---------------	---

---

Function Description	Indicate the status of Transmit interrupt flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>RESET: When next data is written in Transmit data register.</li> <li>SET: When Transmit data register is empty.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR TXIS LL_I2C_IsActiveFlag_TXIS</li> </ul>

### LL\_I2C\_IsActiveFlag\_RXNE

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE( I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Receive data register not empty flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>RESET: When Receive data register is read.</li> <li>SET: When the received data is copied in Receive data register.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR RXNE LL_I2C_IsActiveFlag_RXNE</li> </ul>

### LL\_I2C\_IsActiveFlag\_ADDR

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR( I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Address matched flag (slave mode).
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>RESET: Clear default value.</li> <li>SET: When the received slave address matched with one of the enabled slave address.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR ADDR LL_I2C_IsActiveFlag_ADDR</li> </ul>

### LL\_I2C\_IsActiveFlag\_NACK

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK( I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Not Acknowledge received flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>RESET: Clear default value.</li> <li>SET: When a NACK is received after a byte transmission.</li> </ul>

- Reference Manual to LL API cross reference:
- ISR NACKF LL\_I2C\_IsActiveFlag\_NACK

### **LL\_I2C\_IsActiveFlag\_STOP**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP(I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Stop detection flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When a Stop condition is detected.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR STOPF LL_I2C_IsActiveFlag_STOP</li> </ul>

### **LL\_I2C\_IsActiveFlag\_TC**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC(I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Transfer complete flag (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TC LL_I2C_IsActiveFlag_TC</li> </ul>

### **LL\_I2C\_IsActiveFlag\_TCR**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR(I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of Transfer complete flag (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TCR LL_I2C_IsActiveFlag_TCR</li> </ul>

### **LL\_I2C\_IsActiveFlag\_BERR**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR</code>
---------------	--

**(I2C\_TypeDef \* I2Cx)**

Function Description	Indicate the status of Bus error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR BERR LL_I2C_IsActiveFlag_BERR</li> </ul>

**LL\_I2C\_IsActiveFlag\_ARLO**

Function Name	<b>_STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)</b>
Function Description	Indicate the status of Arbitration lost flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When arbitration lost.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ARLO LL_I2C_IsActiveFlag_ARLO</li> </ul>

**LL\_I2C\_IsActiveFlag\_OVR**

Function Name	<b>_STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)</b>
Function Description	Indicate the status of Overrun/Underrun flag (slave mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR OVR LL_I2C_IsActiveFlag_OVR</li> </ul>

**LL\_I2C\_IsActiveFlag\_BUSY**

Function Name	<b>_STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)</b>
Function Description	Indicate the status of Bus Busy flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Clear default value. SET: When a Start condition is detected.</li> </ul>

- Reference Manual to • ISR BUSY LL\_I2C\_IsActiveFlag\_BUSY  
LL API cross reference:

### 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:**

- Reference Manual to • ICR ADDRCF LL\_I2C\_ClearFlag\_ADDR  
LL API cross reference:

### LL\_I2C\_ClearFlag\_NACK

Function Name **\_STATIC\_INLINE void LL\_I2C\_ClearFlag\_NACK (I2C\_TypeDef \* I2Cx)**

Function Description Clear Not Acknowledge flag.

Parameters • **I2Cx:** I2C Instance.

Return values • **None:**

- Reference Manual to • ICR NACKCF LL\_I2C\_ClearFlag\_NACK  
LL API cross reference:

### 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:**

- Reference Manual to • ICR STOPCF LL\_I2C\_ClearFlag\_STOP  
LL API cross reference:

### LL\_I2C\_ClearFlag\_TXE

Function Name **\_STATIC\_INLINE void LL\_I2C\_ClearFlag\_TXE (I2C\_TypeDef \* I2Cx)**

Function Description Clear Transmit data register empty flag (TXE).

Parameters • **I2Cx:** I2C Instance.

Return values • **None:**

Notes • This bit can be clear by software in order to flush the transmit

data register (TXDR).

Reference Manual to  
LL API cross  
reference:

- ISR TXE LL\_I2C\_ClearFlag\_TXE

### **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  
LL API cross  
reference:

- ICR BERRCF LL\_I2C\_ClearFlag\_BERR

### **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:

- ICR ARLOCF 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:

- ICR OVRCF LL\_I2C\_ClearFlag\_OVR

### **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	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR</li> </ul>

### LL\_I2C\_IsActiveSMBusFlag\_TIMEOUT

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of SMBus Timeout detection flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT</li> </ul>

### LL\_I2C\_IsActiveSMBusFlag\_ALERT

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)</code>
Function Description	Indicate the status of SMBus alert flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT</li> </ul>

### LL\_I2C\_ClearSMBusFlag\_PECERR

Function Name	<code>__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)</code>
Function Description	Clear SMBus PEC error flag.

---

Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ICR PECCF LL_I2C_ClearSMBusFlag_PECERR</li> </ul>

### LL\_I2C\_ClearSMBusFlag\_TIMEOUT

Function Name	<code>_STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT(I2C_TypeDef * I2Cx)</code>
Function Description	Clear SMBus Timeout detection flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ICR TIMEOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT</li> </ul>

### LL\_I2C\_ClearSMBusFlag\_ALERT

Function Name	<code>_STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT(I2C_TypeDef * I2Cx)</code>
Function Description	Clear SMBus Alert flag.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT</li> </ul>

### LL\_I2C\_EnableAutoEndMode

Function Name	<code>_STATIC_INLINE void LL_I2C_EnableAutoEndMode(I2C_TypeDef * I2Cx)</code>
Function Description	Enable automatic STOP condition generation (master mode).
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Automatic end mode : a STOP condition is automatically sent</li> </ul>

when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.

Reference Manual to  
LL API cross  
reference:

- CR2 AUTOEND LL\_I2C\_EnableAutoEndMode

### **LL\_I2C\_DisableAutoEndMode**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_DisableAutoEndMode(I2C_TypeDef * I2Cx)</code></b>
Function Description	Disable automatic STOP condition generation (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Software end mode : TC flag is set when NBYTES data are transferred, stretching SCL low.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 AUTOEND LL_I2C_DisableAutoEndMode</li> </ul>

### **LL\_I2C\_IsEnabledAutoEndMode**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode(I2C_TypeDef * I2Cx)</code></b>
Function Description	Check if automatic STOP condition is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode</li> </ul>

### **LL\_I2C\_EnableReloadMode**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_EnableReloadMode(I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable reload mode (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RELOAD LL_I2C_EnableReloadMode</li> </ul>

### **LL\_I2C\_DisableReloadMode**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_DisableReloadMode</code></b>
---------------	--

**(I2C\_TypeDef \* I2Cx)**

Function Description	Disable reload mode (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RELOAD LL_I2C_DisableReloadMode</li> </ul>

**LL\_I2C\_IsEnabledReloadMode**

Function Name	<b>__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (I2C_TypeDef * I2Cx)</b>
Function Description	Check if reload mode is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RELOAD LL_I2C_IsEnabledReloadMode</li> </ul>

**LL\_I2C\_SetTransferSize**

Function Name	<b>__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)</b>
Function Description	Configure the number of bytes for transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>TransferSize:</b> This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Changing these bits when START bit is set is not allowed.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 NBYTES LL_I2C_SetTransferSize</li> </ul>

**LL\_I2C\_GetTransferSize**

Function Name	<b>__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (I2C_TypeDef * I2Cx)</b>
Function Description	Get the number of bytes configured for transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x0 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR2 NBYTES LL_I2C_GetTransferSize</li> </ul>

reference:

### **LL\_I2C\_AcknowledgeNextData**

Function Name	<b>STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)</b>
Function Description	Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>TypeAcknowledge:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_ACK</li> <li>- LL_I2C_NACK</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Usage in Slave mode only.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 NACK LL_I2C_AcknowledgeNextData</li> </ul>

### **LL\_I2C\_GenerateStopCondition**

Function Name	<b>STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)</b>
Function Description	Generate a STOP condition after the current byte transfer (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 STOP LL_I2C_GenerateStopCondition</li> </ul>

### **LL\_I2C\_GenerateStartCondition**

Function Name	<b>STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)</b>
Function Description	Generate a START or RESTART condition.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 START LL_I2C_GenerateStartCondition</li> </ul>

**LL\_I2C\_EnableAuto10BitRead**

Function Name	<code>__STATIC_INLINE void LL_I2C_EnableAuto10BitRead(I2C_TypeDef * I2Cx)</code>
Function Description	Enable automatic RESTART Read request condition for 10bit address header (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 HEAD10R LL_I2C_EnableAuto10BitRead</li> </ul>

**LL\_I2C\_DisableAuto10BitRead**

Function Name	<code>__STATIC_INLINE void LL_I2C_DisableAuto10BitRead(I2C_TypeDef * I2Cx)</code>
Function Description	Disable automatic RESTART Read request condition for 10bit address header (master mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The master only sends the first 7 bits of 10bit address in Read direction.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 HEAD10R LL_I2C_DisableAuto10BitRead</li> </ul>

**LL\_I2C\_IsEnabledAuto10BitRead**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead(I2C_TypeDef * I2Cx)</code>
Function Description	Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead</li> </ul>

**LL\_I2C\_SetTransferRequest**

Function Name	<code>__STATIC_INLINE void LL_I2C_SetTransferRequest(I2C_TypeDef * I2Cx, uint32_t TransferRequest)</code>
Function Description	Configure the transfer direction (master mode).

Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>TransferRequest:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_REQUEST_WRITE</li> <li>– LL_I2C_REQUEST_READ</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Changing these bits when START bit is set is not allowed.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 RD_WRN LL_I2C_SetTransferRequest</li> </ul>

### LL\_I2C\_GetTransferRequest

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (I2C_TypeDef * I2Cx)</code>
Function Description	Get the transfer direction requested (master mode).
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_REQUEST_WRITE</li> <li>– LL_I2C_REQUEST_READ</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 RD_WRN LL_I2C_GetTransferRequest</li> </ul>

### LL\_I2C\_SetSlaveAddr

Function Name	<code>__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)</code>
Function Description	Configure the slave address for transfer (master mode).
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> <li><b>SlaveAddr:</b> This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Changing these bits when START bit is set is not allowed.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 SADD LL_I2C_SetSlaveAddr</li> </ul>

### LL\_I2C\_GetSlaveAddr

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (I2C_TypeDef * I2Cx)</code>
Function Description	Get the slave address programmed for transfer.
Parameters	<ul style="list-style-type: none"> <li><b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x0 and Max_Data=0x3F</li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CR2 SADD LL\_I2C\_GetSlaveAddr

### LL\_I2C\_HandleTransfer

Function Name	<code>_STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)</code>
Function Description	Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>SlaveAddr:</b> Specifies the slave address to be programmed.</li> <li>• <b>SlaveAddrSize:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_ADDRSOLVE_7BIT</li> <li>- LL_I2C_ADDRSOLVE_10BIT</li> </ul> </li> <li>• <b>TransferSize:</b> Specifies the number of bytes to be programmed. This parameter must be a value between 0 and 255.</li> <li>• <b>EndMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_MODE_RELOAD</li> <li>- LL_I2C_MODE_AUTOEND</li> <li>- LL_I2C_MODE_SOFTEND</li> <li>- LL_I2C_MODE_SMBUS_RELOAD</li> <li>- LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC</li> <li>- LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC</li> <li>- LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC</li> <li>- LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC</li> </ul> </li> <li>• <b>Request:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_I2C_GENERATE_NOSTARTSTOP</li> <li>- LL_I2C_GENERATE_STOP</li> <li>- LL_I2C_GENERATE_START_READ</li> <li>- LL_I2C_GENERATE_START_WRITE</li> <li>- LL_I2C_GENERATE_RESTART_7BIT_READ</li> <li>- LL_I2C_GENERATE_RESTART_7BIT_WRITE</li> <li>- LL_I2C_GENERATE_RESTART_10BIT_READ</li> <li>- LL_I2C_GENERATE_RESTART_10BIT_WRITE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 SADD LL_I2C_HandleTransfer</li> <li>• CR2 ADD10 LL_I2C_HandleTransfer</li> <li>• CR2 RD_WRN LL_I2C_HandleTransfer</li> <li>• CR2 START LL_I2C_HandleTransfer</li> <li>• CR2 STOP LL_I2C_HandleTransfer</li> <li>• CR2 RELOAD LL_I2C_HandleTransfer</li> <li>• CR2 NBYTES LL_I2C_HandleTransfer</li> <li>• CR2 AUTOEND LL_I2C_HandleTransfer</li> <li>• CR2 HEAD10R LL_I2C_HandleTransfer</li> </ul>

**LL\_I2C\_GetTransferDirection**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_GetTransferDirection(I2C_TypeDef * I2Cx)</code></b>
Function Description	Indicate the value of transfer direction (slave mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_I2C_DIRECTION_WRITE</li> <li>– LL_I2C_DIRECTION_READ</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR DIR LL_I2C_GetTransferDirection</li> </ul>

**LL\_I2C\_GetAddressMatchCode**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode(I2C_TypeDef * I2Cx)</code></b>
Function Description	Return the slave matched address.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x3F</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ADDCODE LL_I2C_GetAddressMatchCode</li> </ul>

**LL\_I2C\_EnableSMBusPECCompare**

Function Name	<b><code>_STATIC_INLINE void LL_I2C_EnableSMBusPECCompare(I2C_TypeDef * I2Cx)</code></b>
Function Description	Enable internal comparison of Packet Error byte (transmission or reception mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> <li>• This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 PECBYTE LL_I2C_EnableSMBusPECCompare</li> </ul>

**LL\_I2C\_IsEnabledSMBusPECCompare**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (I2C_TypeDef * I2Cx)</code>
Function Description	Check if Packet Error byte internal comparison is requested or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare</li> </ul>

**LL\_I2C\_GetSMBusPEC**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)</code>
Function Description	Get the Packet Error byte calculated.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMBUS_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PECR PEC LL_I2C_GetSMBusPEC</li> </ul>

**LL\_I2C\_ReceiveData8**

Function Name	<code>__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)</code>
Function Description	Read Receive Data register.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RXDR RXDATA LL_I2C_ReceiveData8</li> </ul>

**LL\_I2C\_TransmitData8**

Function Name	<code>__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)</code>
Function Description	Write in Transmit Data Register .
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>Data:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TXDR TXDATA LL_I2C_TransmitData8</li> </ul>

### LL\_I2C\_Init

Function Name	<code>uint32_t LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)</code>
Function Description	Initialize the I2C registers according to the specified parameters in <code>I2C_InitStruct</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> <li>• <b>I2C_InitStruct:</b> pointer to a <code>LL_I2C_InitTypeDef</code> structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: I2C registers are initialized</li> <li>– ERROR: Not applicable</li> </ul> </li> </ul>

### LL\_I2C\_DeInit

Function Name	<code>uint32_t LL_I2C_DeInit (I2C_TypeDef * I2Cx)</code>
Function Description	De-initialize the I2C registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> I2C Instance.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: I2C registers are de-initialized</li> <li>– ERROR: I2C registers are not de-initialized</li> </ul> </li> </ul>

### LL\_I2C\_StructInit

Function Name	<code>void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)</code>
Function Description	Set each <code>LL_I2C_InitTypeDef</code> field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2C_InitStruct:</b> Pointer to a <code>LL_I2C_InitTypeDef</code> structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## 63.3 I2C Firmware driver defines

### 63.3.1 I2C

#### *Master Addressing Mode*

`LL_I2C_ADDRESSING_MODE_7BIT` Master operates in 7-bit addressing mode.

`LL_I2C_ADDRESSING_MODE_10BIT` Master operates in 10-bit addressing mode.

#### *Slave Address Length*

`LL_I2C_ADDRSOLVE_7BIT` Slave Address in 7-bit.

`LL_I2C_ADDRSOLVE_10BIT` Slave Address in 10-bit.

#### *Analog Filter Selection*

`LL_I2C_ANALOGFILTER_ENABLE` Analog filter is enabled.  
`LL_I2C_ANALOGFILTER_DISABLE` Analog filter is disabled.

#### ***Clear Flags Defines***

<code>LL_I2C_ICR_ADDRCF</code>	Address Matched flag
<code>LL_I2C_ICR_NACKCF</code>	Not Acknowledge flag
<code>LL_I2C_ICR_STOPCF</code>	Stop detection flag
<code>LL_I2C_ICR_BERRCF</code>	Bus error flag
<code>LL_I2C_ICR_ARLOCF</code>	Arbitration Lost flag
<code>LL_I2C_ICR_OVRCF</code>	Overrun/Underrun flag
<code>LL_I2C_ICR_PECCF</code>	PEC error flag
<code>LL_I2C_ICR_TIMOUTCF</code>	Timeout detection flag
<code>LL_I2C_ICR_ALERTCF</code>	Alert flag

#### ***Read Write Direction***

<code>LL_I2C_DIRECTION_WRITE</code>	Write transfer request by master, slave enters receiver mode.
<code>LL_I2C_DIRECTION_READ</code>	Read transfer request by master, slave enters transmitter mode.

#### ***DMA Register Data***

<code>LL_I2C_DMA_REG_DATA_TRANSMIT</code>	Get address of data register used for transmission
<code>LL_I2C_DMA_REG_DATA_RECEIVE</code>	Get address of data register used for reception

#### ***Start And Stop Generation***

<code>LL_I2C_GENERATE_NOSTARTSTOP</code>	Don't Generate Stop and Start condition.
<code>LL_I2C_GENERATE_STOP</code>	Generate Stop condition (Size should be set to 0).
<code>LL_I2C_GENERATE_START_READ</code>	Generate Start for read request.
<code>LL_I2C_GENERATE_START_WRITE</code>	Generate Start for write request.
<code>LL_I2C_GENERATE_RESTART_7BIT_READ</code>	Generate Restart for read request, slave 7Bit address.
<code>LL_I2C_GENERATE_RESTART_7BIT_WRITE</code>	Generate Restart for write request, slave 7Bit address.
<code>LL_I2C_GENERATE_RESTART_10BIT_READ</code>	Generate Restart for read request, slave 10Bit address.
<code>LL_I2C_GENERATE_RESTART_10BIT_WRITE</code>	Generate Restart for write request, slave 10Bit address.

#### ***Get Flags Defines***

<code>LL_I2C_ISR_TXE</code>	Transmit data register empty
<code>LL_I2C_ISR_TXIS</code>	Transmit interrupt status

LL_I2C_ISR_RXNE	Receive data register not empty
LL_I2C_ISR_ADDR	Address matched (slave mode)
LL_I2C_ISR_NACKF	Not Acknowledge received flag
LL_I2C_ISR_STOPF	Stop detection flag
LL_I2C_ISR_TC	Transfer Complete (master mode)
LL_I2C_ISR_TCR	Transfer Complete Reload
LL_I2C_ISR_BERR	Bus error
LL_I2C_ISR_ARLO	Arbitration lost
LL_I2C_ISR_OVR	Overrun/Underrun (slave mode)
LL_I2C_ISR_PECERR	PEC Error in reception (SMBus mode)
LL_I2C_ISR_TIMEOUT	Timeout detection flag (SMBus mode)
LL_I2C_ISR_ALERT	SMBus alert (SMBus mode)
LL_I2C_ISR_BUSY	Bus busy

#### 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_CR1_TXIE	TX Interrupt enable
LL_I2C_CR1_RXIE	RX Interrupt enable
LL_I2C_CR1_ADDRIE	Address match Interrupt enable (slave only)
LL_I2C_CR1_NACKIE	Not acknowledge received Interrupt enable
LL_I2C_CR1_STOPIE	STOP detection Interrupt enable
LL_I2C_CR1_TCIE	Transfer Complete interrupt enable
LL_I2C_CR1_ERRIE	Error interrupts enable

#### Transfer End Mode

LL_I2C_MODE_RELOAD	Enable I2C Reload mode.
LL_I2C_MODE_AUTOEND	Enable I2C Automatic end mode with no HW PEC comparison.
LL_I2C_MODE_SOFTEND	Enable I2C Software end mode with no HW PEC comparison.
LL_I2C_MODE_SMBUS_RELOAD	Enable SMBUS Automatic end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC	Enable SMBUS Automatic end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC	Enable SMBUS Software end mode with HW PEC comparison.
LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC	Enable SMBUS Automatic end mode with HW PEC comparison.

`LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC` Enable SMBUS Software end mode with HW PEC comparison.

#### ***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.

#### ***Own Address 2 Masks***

`LL_I2C_OWNADDRESS2_NOMASK` Own Address2 No mask.

`LL_I2C_OWNADDRESS2_MASK01` Only Address2 bits[7:2] are compared.

`LL_I2C_OWNADDRESS2_MASK02` Only Address2 bits[7:3] are compared.

`LL_I2C_OWNADDRESS2_MASK03` Only Address2 bits[7:4] are compared.

`LL_I2C_OWNADDRESS2_MASK04` Only Address2 bits[7:5] are compared.

`LL_I2C_OWNADDRESS2_MASK05` Only Address2 bits[7:6] are compared.

`LL_I2C_OWNADDRESS2_MASK06` Only Address2 bits[7] are compared.

`LL_I2C_OWNADDRESS2_MASK07` No comparison is done. All Address2 are acknowledged.

#### ***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

#### ***Transfer Request Direction***

`LL_I2C_REQUEST_WRITE` Master request a write transfer.

`LL_I2C_REQUEST_READ` Master request a read transfer.

#### ***SMBus TimeoutA Mode SCL SDA Timeout***

`LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW` TimeoutA is used to detect SCL low level timeout.

`LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH` TimeoutA is used to detect both SCL and SDA high level timeout.

#### ***SMBus Timeout Selection***

`LL_I2C_SMBUS_TIMEOUTA` TimeoutA enable bit

`LL_I2C_SMBUS_TIMEOUTB` TimeoutB (extended clock) enable bit

`LL_I2C_SMBUS_ALL_TIMEOUT` TimeoutA and TimeoutB (extended clock) enable bits

#### ***Convert SDA SCL timings***

`_LL_I2C_CONVERT_TIMINGS` **Description:**

- Configure the SDA setup, hold time and the SCL high, low period.

**Parameters:**

- PRESCALER: This parameter must be a value between 0 and 0xF.
- DATA\_SETUP\_TIME: This parameter must be a value between 0 and 0xF. ( $tscldel = (SCLDEL+1)xtpres$ )
- DATA\_HOLD\_TIME: This parameter must be a value between 0 and 0xF. ( $tsdadel = SDADELxtpres$ )
- CLOCK\_HIGH\_PERIOD: This parameter must be a value between 0 and 0xFF. ( $tschl = (SCLH+1)xtpres$ )
- CLOCK\_LOW\_PERIOD: This parameter must be a value between 0 and 0xFF. ( $tscli = (SCLL+1)xtpres$ )

**Return value:**

- Value: between 0 and 0xFFFFFFFF

**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

## 64 LL I2S Generic Driver

### 64.1 I2S Firmware driver registers structures

#### 64.1.1 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()**.

## 64.2 I2S Firmware driver API description

### 64.2.1 Detailed description of functions

#### LL\_I2S\_Enable

Function Name	<code>__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)</code>
Function Description	Select I2S mode and Enable I2S peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SCFGR I2SMOD LL_I2S_Enable</li> <li>• I2SCFGR I2SE LL_I2S_Enable</li> </ul>

#### LL\_I2S\_Disable

Function Name	<code>__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)</code>
Function Description	Disable I2S peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SCFGR I2SE LL_I2S_Disable</li> </ul>

#### LL\_I2S\_IsEnabled

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsEnabled (SPI_TypeDef * SPIx)</code>
Function Description	Check if I2S peripheral is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SCFGR I2SE LL_I2S_IsEnabled</li> </ul>

#### LL\_I2S\_SetDataFormat

Function Name	<code>__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataFormat)</code>
Function Description	Set I2S Data frame length.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>DataFormat:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_I2S_DATAFORMAT_16B</li> <li>– LL_I2S_DATAFORMAT_16B_EXTENDED</li> <li>– LL_I2S_DATAFORMAT_24B</li> </ul> </li> </ul>

- 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\_SetDataFormat
- I2SCFGR CHLEN LL\_I2S\_SetDataFormat

### **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

- I2SCFGR CKPOL LL\_I2S\_SetClockPolarity

reference:

### **LL\_I2S\_SetStandard**

Function Name	<b><code>__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)</code></b>
Function Description	Set I2S Standard Protocol.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>Standard:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_I2S_STANDARD_PHILIPS</code></li> <li>- <code>LL_I2S_STANDARD_MSB</code></li> <li>- <code>LL_I2S_STANDARD_LSB</code></li> <li>- <code>LL_I2S_STANDARD_PCM_SHORT</code></li> <li>- <code>LL_I2S_STANDARD_PCM_LONG</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SCFGR I2SSTD LL_I2S_SetStandard</li> <li>• I2SCFGR PCMSYNC LL_I2S_SetStandard</li> </ul>

### **LL\_I2S\_GetStandard**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)</code></b>
Function Description	Get I2S Standard Protocol.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_I2S_STANDARD_PHILIPS</code></li> <li>- <code>LL_I2S_STANDARD_MSB</code></li> <li>- <code>LL_I2S_STANDARD_LSB</code></li> <li>- <code>LL_I2S_STANDARD_PCM_SHORT</code></li> <li>- <code>LL_I2S_STANDARD_PCM_LONG</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SCFGR I2SSTD LL_I2S_SetStandard</li> <li>• I2SCFGR PCMSYNC LL_I2S_SetStandard</li> </ul>

### **LL\_I2S\_SetTransferMode**

Function Name	<b><code>__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)</code></b>
Function Description	Set I2S Transfer Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>Mode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_I2S_MODE_SLAVE_TX</code></li> <li>- <code>LL_I2S_MODE_SLAVE_RX</code></li> <li>- <code>LL_I2S_MODE_MASTER_TX</code></li> <li>- <code>LL_I2S_MODE_MASTER_RX</code></li> </ul> </li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>I2SCFGR I2SCFG LL_I2S_SetTransferMode</li> </ul>

### LL\_I2S\_SetTransferMode

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_SetTransferMode(SPI_TypeDef * SPIx)</code>
Function Description	Get I2S Transfer Mode.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>LL_I2S_MODE_SLAVE_TX</li> <li>LL_I2S_MODE_SLAVE_RX</li> <li>LL_I2S_MODE_MASTER_TX</li> <li>LL_I2S_MODE_MASTER_RX</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>I2SCFGR I2SCFG LL_I2S_SetTransferMode</li> </ul>

### LL\_I2S\_SetPrescalerLinear

Function Name	<code>__STATIC_INLINE void LL_I2S_SetPrescalerLinear(SPI_TypeDef * SPIx, uint8_t PrescalerLinear)</code>
Function Description	Set I2S linear prescaler.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>PrescalerLinear:</b> Value between Min_Data=0x02 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>I2SPR I2SDIV LL_I2S_SetPrescalerLinear</li> </ul>

### LL\_I2S\_GetPrescalerLinear

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear(SPI_TypeDef * SPIx)</code>
Function Description	Get I2S linear prescaler.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>PrescalerLinear:</b> Value between Min_Data=0x02 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>I2SPR I2SDIV LL_I2S_SetPrescalerLinear</li> </ul>

**LL\_I2S\_SetPrescalerParity**

Function Name	<code>__STATIC_INLINE void LL_I2S_SetPrescalerParity(SPI_TypeDef * SPIx, uint32_t PrescalerParity)</code>
Function Description	Set I2S parity prescaler.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>PrescalerParity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_I2S_PRESCALER_PARITY EVEN</code></li> <li>- <code>LL_I2S_PRESCALER_PARITY ODD</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SPR ODD LL_I2S_SetPrescalerParity</li> </ul>

**LL\_I2S\_GetPrescalerParity**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity(SPI_TypeDef * SPIx)</code>
Function Description	Get I2S parity prescaler.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_I2S_PRESCALER_PARITY EVEN</code></li> <li>- <code>LL_I2S_PRESCALER_PARITY ODD</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SPR ODD LL_I2S_GetPrescalerParity</li> </ul>

**LL\_I2S\_EnableMasterClock**

Function Name	<code>__STATIC_INLINE void LL_I2S_EnableMasterClock(SPI_TypeDef * SPIx)</code>
Function Description	Enable the Master Clock Output (Pin MCK)
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• I2SPR MCKOE LL_I2S_EnableMasterClock</li> </ul>

**LL\_I2S\_DisableMasterClock**

Function Name	<code>__STATIC_INLINE void LL_I2S_DisableMasterClock(SPI_TypeDef * SPIx)</code>
Function Description	Disable the Master Clock Output (Pin MCK)
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to I2SPR MCKOE LL\_I2S\_DisableMasterClock  
LL API cross reference:
- I2SPR MCKOE LL\_I2S\_DisableMasterClock

### **LL\_I2S\_IsEnabledMasterClock**

Function Name	<code>_STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock(SPI_TypeDef * SPIx)</code>
Function Description	Check if the Master Clock Ouput (Pin MCK) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

Reference Manual to I2SPR MCKOE LL\_I2S\_IsEnabledMasterClock  
LL API cross reference:

### **LL\_I2S\_EnableAsyncStart**

Function Name	<code>_STATIC_INLINE void LL_I2S_EnableAsyncStart(SPI_TypeDef * SPIx)</code>
Function Description	Enable Asynchronous Start.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to I2SCFGR ASTRTEN LL\_I2S\_EnableAsyncStart  
LL API cross reference:

### **LL\_I2S\_DisableAsyncStart**

Function Name	<code>_STATIC_INLINE void LL_I2S_DisableAsyncStart(SPI_TypeDef * SPIx)</code>
Function Description	Disable Asynchronous Start.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to I2SCFGR ASTRTEN LL\_I2S\_DisableAsyncStart  
LL API cross reference:

### **LL\_I2S\_IsEnabledAsyncStart**

Function Name	<code>_STATIC_INLINE uint32_t LL_I2S_IsEnabledAsyncStart(SPI_TypeDef * SPIx)</code>
Function Description	Check if Asynchronous Start is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

Reference Manual to I2SCFGR ASTRTEN LL\_I2S\_IsEnabledAsyncStart  
LL API cross

reference:

### LL\_I2S\_IsActiveFlag\_RXNE

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE(SPI_TypeDef * SPIx)</code>
Function Description	Check if Rx buffer is not empty.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>SR RXNE LL_I2S_IsActiveFlag_RXNE</li></ul>

### LL\_I2S\_IsActiveFlag\_TXE

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE(SPI_TypeDef * SPIx)</code>
Function Description	Check if Tx buffer is empty.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>SR TXE LL_I2S_IsActiveFlag_TXE</li></ul>

### LL\_I2S\_IsActiveFlag\_BSY

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY(SPI_TypeDef * SPIx)</code>
Function Description	Get Busy flag.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>SR BSY LL_I2S_IsActiveFlag_BSY</li></ul>

### LL\_I2S\_IsActiveFlag\_OVR

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR(SPI_TypeDef * SPIx)</code>
Function Description	Get Overrun error flag.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>SR OVR LL_I2S_IsActiveFlag_OVR</li></ul>

**LL\_I2S\_IsActiveFlag\_UDR**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (SPI_TypeDef * SPIx)</code>
Function Description	Get Underrun error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR UDR LL_I2S_IsActiveFlag_UDR</li> </ul>

**LL\_I2S\_IsActiveFlag\_FRE**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (SPI_TypeDef * SPIx)</code>
Function Description	Get Frame format error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR FRE LL_I2S_IsActiveFlag_FRE</li> </ul>

**LL\_I2S\_IsActiveFlag\_CHSIDE**

Function Name	<code>__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_CHSIDE (SPI_TypeDef * SPIx)</code>
Function Description	Get Channel side flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE</li> </ul>

**LL\_I2S\_ClearFlag\_OVR**

Function Name	<code>__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)</code>
Function Description	Clear Overrun error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• SR OVR LL_I2S_ClearFlag_OVR</li> </ul>

reference:

### **LL\_I2S\_ClearFlag\_UDR**

Function Name	<code>_STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)</code>
Function Description	Clear Underrun error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR UDR LL_I2S_ClearFlag_UDR</li> </ul>

### **LL\_I2S\_ClearFlag\_FRE**

Function Name	<code>_STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)</code>
Function Description	Clear Frame format error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR FRE LL_I2S_ClearFlag_FRE</li> </ul>

### **LL\_I2S\_EnableIT\_ERR**

Function Name	<code>_STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)</code>
Function Description	Enable error IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ERRIE LL_I2S_EnableIT_ERR</li> </ul>

### **LL\_I2S\_EnableIT\_RXNE**

Function Name	<code>_STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)</code>
Function Description	Enable Rx buffer not empty IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CR2 RXNEIE LL_I2S_EnableIT_RXNE</li> </ul>

LL API cross  
reference:

### **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:

### **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:

### **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:

### **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 • CR2 TXEIE LL\_I2S\_DisableIT\_TXE  
LL API cross reference:

### **LL\_I2S\_IsEnabledIT\_ERR**

Function Name	<code>_STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_ERR(SPI_TypeDef * SPIx)</code>
Function Description	Check if ERR IT is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CR2 ERRIE LL_I2S_IsEnabledIT_ERR

### **LL\_I2S\_IsEnabledIT\_RXNE**

Function Name	<code>_STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXNE(SPI_TypeDef * SPIx)</code>
Function Description	Check if RXNE IT is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CR2 RXNEIE LL_I2S_IsEnabledIT_RXNE

### **LL\_I2S\_IsEnabledIT\_TXE**

Function Name	<code>_STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXE(SPI_TypeDef * SPIx)</code>
Function Description	Check if TXE IT is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CR2 TXEIE LL_I2S_IsEnabledIT_TXE

### **LL\_I2S\_EnableDMAReq\_RX**

Function Name	<code>_STATIC_INLINE void LL_I2S_EnableDMAReq_RX(SPI_TypeDef * SPIx)</code>
Function Description	Enable DMA Rx.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	• CR2 RXDMAEN LL_I2S_EnableDMAReq_RX

reference:

### **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	<code>_STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX(SPI_TypeDef * SPIx)</code>
Function Description	Check if DMA Tx is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TXDMAEN LL_I2S_IsEnabledDMAReq_TX</li> </ul>

**LL\_I2S\_ReceiveData16**

Function Name	<code>_STATIC_INLINE uint16_t LL_I2S_ReceiveData16(SPI_TypeDef * SPIx)</code>
Function Description	Read 16-Bits in data register.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>RxDATA:</b> Value between Min_Data=0x0000 and Max_Data=0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DR LL_I2S_ReceiveData16</li> </ul>

**LL\_I2S\_TransmitData16**

Function Name	<code>_STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)</code>
Function Description	Write 16-Bits in data register.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>TxData:</b> Value between Min_Data=0x0000 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DR LL_I2S_TransmitData16</li> </ul>

**LL\_I2S\_DeInit**

Function Name	<code>ErrorStatus LL_I2S_DeInit (SPI_TypeDef * SPIx)</code>
Function Description	De-initialize the SPI/I2S registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: SPI registers are de-initialized</li> <li>– ERROR: SPI registers are not de-initialized</li> </ul> </li> </ul>

**LL\_I2S\_Init**

Function Name	<b>ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)</b>
Function Description	Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>I2S_InitStruct:</b> pointer to a LL_I2S_InitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:             <ul style="list-style-type: none"> <li>– SUCCESS: SPI registers are Initialized</li> <li>– ERROR: SPI registers are not Initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

**LL\_I2S\_StructInit**

Function Name	<b>void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)</b>
Function Description	Set each LL_I2S_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2S_InitStruct:</b> pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_I2S\_ConfigPrescaler**

Function Name	<b>void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)</b>
Function Description	Set linear and parity prescaler.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>PrescalerLinear:</b> value: Min_Data=0x02 and Max_Data=0xFF.</li> <li>• <b>PrescalerParity:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_I2S_PRESCALER_PARITY_EVEN</li> <li>– LL_I2S_PRESCALER_PARITY_ODD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).</li> </ul>

**64.3 I2S Firmware driver defines****64.3.1 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

## 65 LL IWDG Generic Driver

### 65.1 IWDG Firmware driver API description

#### 65.1.1 Detailed description of functions

##### LL\_IWDG\_Enable

Function Name **`__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)`**

Function Description Start the Independent Watchdog.

- **IWDGx:** IWDG Instance

Return values **None:**

Notes Except if the hardware watchdog option is selected

Reference Manual to **KR KEY LL\_IWDG\_Enable**

LL API cross reference:

##### 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.

- **IWDGx:** IWDG Instance

Return values **None:**

Reference Manual to **KR KEY LL\_IWDG\_ReloadCounter**

LL API cross

reference:

##### 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.

- **IWDGx:** IWDG Instance

Return values **None:**

Reference Manual to **KR KEY LL\_IWDG\_EnableWriteAccess**

LL API cross

reference:

##### 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	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• KR KEY LL_IWDG_DisableWriteAccess</li> </ul>

**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	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> <li>• <b>Prescaler:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_IWDG_PRESCALER_4</li> <li>- LL_IWDG_PRESCALER_8</li> <li>- LL_IWDG_PRESCALER_16</li> <li>- LL_IWDG_PRESCALER_32</li> <li>- LL_IWDG_PRESCALER_64</li> <li>- LL_IWDG_PRESCALER_128</li> <li>- LL_IWDG_PRESCALER_256</li> </ul> </li> </ul>
------------	---

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

Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PR PR LL_IWDG_SetPrescaler</li> </ul>
---	--

**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	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_IWDG_PRESCALER_4</li> <li>- LL_IWDG_PRESCALER_8</li> <li>- LL_IWDG_PRESCALER_16</li> <li>- LL_IWDG_PRESCALER_32</li> <li>- LL_IWDG_PRESCALER_64</li> <li>- LL_IWDG_PRESCALER_128</li> <li>- LL_IWDG_PRESCALER_256</li> </ul> </li> </ul>

Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PR PR LL_IWDG_GetPrescaler</li> </ul>
---	--

**LL\_IWDG\_SetReloadCounter**

Function Name	<code>_STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)</code>
Function Description	Specify the IWDG down-counter reload value.
Parameters	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> <li>• <b>Counter:</b> Value between Min_Data=0 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RLR RL LL_IWDG_SetReloadCounter</li> </ul>

**LL\_IWDG\_GetReloadCounter**

Function Name	<code>_STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)</code>
Function Description	Get the specified IWDG down-counter reload value.
Parameters	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0 and Max_Data=0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RLR RL LL_IWDG_GetReloadCounter</li> </ul>

**LL\_IWDG\_SetWindow**

Function Name	<code>_STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)</code>
Function Description	Specify high limit of the window value to be compared to the down-counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> <li>• <b>Window:</b> Value between Min_Data=0 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• WINR WIN LL_IWDG_SetWindow</li> </ul>

**LL\_IWDG\_GetWindow**

Function Name	<code>_STATIC_INLINE uint32_t LL_IWDG_GetWindow (IWDG_TypeDef * IWDGx)</code>
Function Description	Get the high limit of the window value specified.
Parameters	<ul style="list-style-type: none"> <li>• <b>IWDGx:</b> IWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0 and Max_Data=0xFFFF</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• WINR WIN LL_IWDG_GetWindow</li> </ul>

LL API cross  
reference:

### **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:

### **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:

### **LL\_IWDG\_IsActiveFlag\_WVU**

Function Name      **STATIC\_INLINE uint32\_t LL\_IWDG\_IsActiveFlag\_WVU(IWDG\_TypeDef \* IWDGx)**

Function Description      Check if flag Window 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:

### **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

• SR PVU LL\_IWDG\_IsReady

• SR WVU LL\_IWDG\_IsReady



- 
- reference:
  - SR RVU LL\_IWDG\_IsReady

## 65.2 IWDG Firmware driver defines

### 65.2.1 IWDG

#### *Get Flags Defines*

LL_IWDG_SR_PVU	Watchdog prescaler value update
LL_IWDG_SR_RVU	Watchdog counter reload value update
LL_IWDG_SR_WVU	Watchdog counter window 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

## 66 LL LPTIM Generic Driver

### 66.1 LPTIM Firmware driver registers structures

#### 66.1.1 LL\_LPTIM\_InitTypeDef

##### Data Fields

- *uint32\_t ClockSource*
- *uint32\_t Prescaler*
- *uint32\_t Waveform*
- *uint32\_t Polarity*

##### Field Documentation

- ***uint32\_t LL\_LPTIM\_InitTypeDef::ClockSource***  
Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of [\*LPTIM\\_LL\\_EC\\_CLK\\_SOURCE\*](#). This feature can be modified afterwards using unitary function [\*LL\\_LPTIM\\_SetClockSource\(\)\*](#).
- ***uint32\_t LL\_LPTIM\_InitTypeDef::Prescaler***  
Specifies the prescaler division ratio. This parameter can be a value of [\*LPTIM\\_LL\\_EC\\_PRESCALER\*](#). This feature can be modified afterwards using unitary function [\*LL\\_LPTIM\\_SetPrescaler\(\)\*](#).
- ***uint32\_t LL\_LPTIM\_InitTypeDef::Waveform***  
Specifies the waveform shape. This parameter can be a value of [\*LPTIM\\_LL\\_EC\\_OUTPUT\\_WAVEFORM\*](#). This feature can be modified afterwards using unitary function [\*LL\\_LPTIM\\_ConfigOutput\(\)\*](#).
- ***uint32\_t LL\_LPTIM\_InitTypeDef::Polarity***  
Specifies waveform polarity. This parameter can be a value of [\*LPTIM\\_LL\\_EC\\_OUTPUT\\_POLARITY\*](#). This feature can be modified afterwards using unitary function [\*LL\\_LPTIM\\_ConfigOutput\(\)\*](#).

### 66.2 LPTIM Firmware driver API description

#### 66.2.1 Detailed description of functions

##### LL\_LPTIM\_Enable

Function Name	<code>_STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Enable the LPTIM instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CR ENABLE LL_LPTIM_Enable</li> </ul>

LL API cross  
reference:

### **LL\_LPTIM\_Disable**

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Disable the LPTIM instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ENABLE LL_LPTIM_Disable</li> </ul>

### **LL\_LPTIM\_IsEnabled**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Indicates whether the LPTIM instance is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ENABLE LL_LPTIM_IsEnabled</li> </ul>

### **LL\_LPTIM\_StartCounter**

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)</code></b>
Function Description	Starts the LPTIM counter in the desired mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>OperatingMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_LPTIM_OPERATING_MODE_CONTINUOUS</code></li> <li>- <code>LL_LPTIM_OPERATING_MODE_ONESHOT</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• LPTIM instance must be enabled before starting the counter.</li> <li>• It is possible to change on the fly from One Shot mode to Continuous mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR CNTSTRT LL_LPTIM_StartCounter</li> <li>• CR SNGSTRT LL_LPTIM_StartCounter</li> </ul>

### **LL\_LPTIM\_SetUpdateMode**

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)</code></b>
---------------	--

Function Description	Set the LPTIM registers update mode (enable/disable register preload)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>UpdateMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_UPDATE_MODE_IMMEDIATE</li> <li>– LL_LPTIM_UPDATE_MODE_ENDOFPERIOD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR PRELOAD LL_LPTIM_SetUpdateMode</li> </ul>

### LL\_LPTIM\_GetUpdateMode

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Get the LPTIM registers update mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_UPDATE_MODE_IMMEDIATE</li> <li>– LL_LPTIM_UPDATE_MODE_ENDOFPERIOD</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR PRELOAD LL_LPTIM_GetUpdateMode</li> </ul>

### LL\_LPTIM\_SetAutoReload

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_SetAutoReload(LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)</code></b>
Function Description	Set the auto reload value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>AutoReload:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The LPTIMx_ARR register content must only be modified when the LPTIM is enabled</li> <li>• After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag be set, will lead to unpredictable results.</li> <li>• autoreload value be strictly greater than the compare value.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ARR ARR LL_LPTIM_SetAutoReload</li> </ul>

**LL\_LPTIM\_GetAutoReload**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Get actual auto reload value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>AutoReload:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ARR ARR LL_LPTIM_GetAutoReload</li> </ul>

**LL\_LPTIM\_SetCompare**

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_SetCompare(LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)</code></b>
Function Description	Set the compare value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>CompareValue:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After a write to the LPTIMx_CMP register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMPOK flag be set, will lead to unpredictable results.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CMP CMP LL_LPTIM_SetCompare</li> </ul>

**LL\_LPTIM\_GetCompare**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPTIM_GetCompare(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Get actual compare value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>CompareValue:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CMP CMP LL_LPTIM_GetCompare</li> </ul>

**LL\_LPTIM\_GetCounter**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPTIM_GetCounter(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Get actual counter value.

Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Counter:</b> value</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CNT CNT LL_LPTIM_GetCounter</li> </ul>

### LL\_LPTIM\_SetCounterMode

Function Name	<code>__STATIC_INLINE void LL_LPTIM_SetCounterMode(LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)</code>
Function Description	Set the counter mode (selection of the LPTIM counter clock source).
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> <li><b>CounterMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_COUNTER_MODE_INTERNAL</li> <li>– LL_LPTIM_COUNTER_MODE_EXTERNAL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>The counter mode can be set only when the LPTIM instance is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR COUNTMODE LL_LPTIM_SetCounterMode</li> </ul>

### LL\_LPTIM\_GetCounterMode

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get the counter mode.
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_COUNTER_MODE_INTERNAL</li> <li>– LL_LPTIM_COUNTER_MODE_EXTERNAL</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR COUNTMODE LL_LPTIM_GetCounterMode</li> </ul>

### LL\_LPTIM\_ConfigOutput

Function Name	<code>__STATIC_INLINE void LL_LPTIM_ConfigOutput(LPTIM_TypeDef * LPTIMx, uint32_t Waveform, uint32_t Polarity)</code>
---------------	---

Function Description	Configure the LPTIM instance output (LPTIMx_OUT)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>Waveform:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_OUTPUT_WAVEFORM_PWM</li> <li>- LL_LPTIM_OUTPUT_WAVEFORM_SETONCE</li> </ul> </li> <li>• <b>Polarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_OUTPUT_POLARITY_REGULAR</li> <li>- LL_LPTIM_OUTPUT_POLARITY_INVERSE</li> </ul> </li> </ul>
Return values	• <b>None:</b>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> <li>• Regarding the LPTIM output polarity the change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR WAVE LL_LPTIM_ConfigOutput</li> <li>• CFGR WAVPOL LL_LPTIM_ConfigOutput</li> </ul>

### LL\_LPTIM\_SetWaveform

Function Name	<code>__STATIC_INLINE void LL_LPTIM_SetWaveform(LPTIM_TypeDef * LPTIMx, uint32_t Waveform)</code>
Function Description	Set waveform shape.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>Waveform:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_OUTPUT_WAVEFORM_PWM</li> <li>- LL_LPTIM_OUTPUT_WAVEFORM_SETONCE</li> </ul> </li> </ul>
Return values	• <b>None:</b>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR WAVE LL_LPTIM_SetWaveform</li> </ul>

### LL\_LPTIM\_GetWaveform

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get actual waveform shape.
Parameters	• <b>LPTIMx:</b> Low-Power Timer instance
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_OUTPUT_WAVEFORM_PWM</li> <li>- LL_LPTIM_OUTPUT_WAVEFORM_SETONCE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR WAVE LL_LPTIM_GetWaveform</li> </ul>

**LL\_LPTIM\_SetPolarity**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_SetPolarity(LPTIM_TypeDef * LPTIMx, uint32_t Polarity)</code>
Function Description	Set output polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>Polarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_OUTPUT_POLARITY_REGULAR</li> <li>– LL_LPTIM_OUTPUT_POLARITY_INVERSE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR WAVPOL LL_LPTIM_SetPolarity</li> </ul>

**LL\_LPTIM\_GetPolarity**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_GetPolarity(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get actual output polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_OUTPUT_POLARITY_REGULAR</li> <li>– LL_LPTIM_OUTPUT_POLARITY_INVERSE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR WAVPOL LL_LPTIM_GetPolarity</li> </ul>

**LL\_LPTIM\_SetPrescaler**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_SetPrescaler(LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)</code>
Function Description	Set actual prescaler division ratio.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>Prescaler:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPTIM_PRESCALER_DIV1</li> <li>– LL_LPTIM_PRESCALER_DIV2</li> <li>– LL_LPTIM_PRESCALER_DIV4</li> <li>– LL_LPTIM_PRESCALER_DIV8</li> <li>– LL_LPTIM_PRESCALER_DIV16</li> <li>– LL_LPTIM_PRESCALER_DIV32</li> <li>– LL_LPTIM_PRESCALER_DIV64</li> <li>– LL_LPTIM_PRESCALER_DIV128</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> <li>• When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be</li> </ul>

updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.

Reference Manual to  
LL API cross  
reference:

- CFGR PRESC LL\_LPTIM\_SetPrescaler

### **LL\_LPTIM\_GetPrescaler**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler( (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get actual prescaler division ratio.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPTIM_PRESCALER_DIV1</li> <li>- LL_LPTIM_PRESCALER_DIV2</li> <li>- LL_LPTIM_PRESCALER_DIV4</li> <li>- LL_LPTIM_PRESCALER_DIV8</li> <li>- LL_LPTIM_PRESCALER_DIV16</li> <li>- LL_LPTIM_PRESCALER_DIV32</li> <li>- LL_LPTIM_PRESCALER_DIV64</li> <li>- LL_LPTIM_PRESCALER_DIV128</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR PRESC LL_LPTIM_SetPrescaler</li> </ul>

### **LL\_LPTIM\_EnableTimeout**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_EnableTimeout( (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Enable the timeout function.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> <li>• The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.</li> <li>• The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR TIMOUT LL_LPTIM_EnableTimeout</li> </ul>

### **LL\_LPTIM\_DisableTimeout**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_DisableTimeout( (LPTIM_TypeDef * LPTIMx)</code>
---------------	---

Function Description	Disable the timeout function.
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function must be called when the LPTIM instance is disabled.</li> <li>A trigger event arriving when the timer is already started will be ignored.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR TIMOUT LL_LPTIM_DisableTimeout</li> </ul>

### LL\_LPTIM\_IsEnabledTimeout

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Indicate whether the timeout function is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR TIMOUT LL_LPTIM_IsEnabledTimeout</li> </ul>

### LL\_LPTIM\_TrigSw

Function Name	<code>__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Start the LPTIM counter.
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function must be called when the LPTIM instance is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGTR TRIGEN LL_LPTIM_TrigSw</li> </ul>

### LL\_LPTIM\_ConfigTrigger

Function Name	<code>__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)</code>
Function Description	Configure the external trigger used as a trigger event for the LPTIM.
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMx:</b> Low-Power Timer instance</li> <li><b>Source:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_LPTIM_TRIG_SOURCE_GPIO</li> <li>LL_LPTIM_TRIG_SOURCE_RTCALARMA</li> </ul> </li> </ul>

- LL\_LPTIM\_TRIG\_SOURCE\_RTCALARMB
  - LL\_LPTIM\_TRIG\_SOURCE\_RTCTAMP1
  - LL\_LPTIM\_TRIG\_SOURCE\_RTCTAMP2
  - LL\_LPTIM\_TRIG\_SOURCE\_RTCTAMP3
  - LL\_LPTIM\_TRIG\_SOURCE\_COMP1
  - LL\_LPTIM\_TRIG\_SOURCE\_COMP2
  - **Filter:** This parameter can be one of the following values:
    - LL\_LPTIM\_TRIG\_FILTER\_NONE
    - LL\_LPTIM\_TRIG\_FILTER\_2
    - LL\_LPTIM\_TRIG\_FILTER\_4
    - LL\_LPTIM\_TRIG\_FILTER\_8
  - **Polarity:** This parameter can be one of the following values:
    - LL\_LPTIM\_POLARITY\_RISING
    - LL\_LPTIM\_POLARITY\_FALLING
    - LL\_LPTIM\_POLARITY\_RISING\_FALLING
- Return values**
- **None:**
- Notes**
- This function must be called when the LPTIM instance is disabled.
  - An internal clock source must be present when a digital filter is required for the trigger.
- Reference Manual to LL API cross reference:**
- CFGR TRIGSEL LL\_LPTIM\_ConfigTrigger
  - CFGR TRGFLT LL\_LPTIM\_ConfigTrigger
  - CFGR TRIGEN LL\_LPTIM\_ConfigTrigger

### LL\_LPTIM\_GetTriggerSource

<b>Function Name</b>	<code>STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource(LPTIM_TypeDef * LPTIMx)</code>
<b>Function Description</b>	Get actual external trigger source.
<b>Parameters</b>	• <b>LPTIMx:</b> Low-Power Timer instance
<b>Return values</b>	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPTIM_TRIG_SOURCE_GPIO</li> <li>- LL_LPTIM_TRIG_SOURCE_RTCALARMA</li> <li>- LL_LPTIM_TRIG_SOURCE_RTCALARMB</li> <li>- LL_LPTIM_TRIG_SOURCE_RTCTAMP1</li> <li>- LL_LPTIM_TRIG_SOURCE_RTCTAMP2</li> <li>- LL_LPTIM_TRIG_SOURCE_RTCTAMP3</li> <li>- LL_LPTIM_TRIG_SOURCE_COMP1</li> <li>- LL_LPTIM_TRIG_SOURCE_COMP2</li> </ul> </li> </ul>
<b>Reference Manual to LL API cross reference:</b>	<ul style="list-style-type: none"> <li>• CFGR TRIGSEL LL_LPTIM_GetTriggerSource</li> </ul>

### LL\_LPTIM\_GetTriggerFilter

<b>Function Name</b>	<code>STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter(LPTIM_TypeDef * LPTIMx)</code>
<b>Function Description</b>	Get actual external trigger filter.

Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_TRIG_FILTER_NONE</li> <li>- LL_LPTIM_TRIG_FILTER_2</li> <li>- LL_LPTIM_TRIG_FILTER_4</li> <li>- LL_LPTIM_TRIG_FILTER_8</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR TRGFLT LL_LPTIM_GetTriggerFilter</li> </ul>

### LL\_LPTIM\_GetTriggerPolarity

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity(     LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get actual external trigger polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_TRIG_POLARITY_RISING</li> <li>- LL_LPTIM_TRIG_POLARITY_FALLING</li> <li>- LL_LPTIM_TRIG_POLARITY_RISING_FALLING</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR TRIGEN LL_LPTIM_GetTriggerPolarity</li> </ul>

### LL\_LPTIM\_SetClockSource

Function Name	<code>__STATIC_INLINE void LL_LPTIM_SetClockSource(     LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)</code>
Function Description	Set the source of the clock used by the LPTIM instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>ClockSource:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPTIM_CLK_SOURCE_INTERNAL</li> <li>- LL_LPTIM_CLK_SOURCE_EXTERNAL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR CKSEL LL_LPTIM_SetClockSource</li> </ul>

### LL\_LPTIM\_GetClockSource

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource(     LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get actual LPTIM instance clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPTIM_CLK_SOURCE_INTERNAL</li> <li>- LL_LPTIM_CLK_SOURCE_EXTERNAL</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR CKSEL LL_LPTIM_GetClockSource</li> </ul>
<b>LL_LPTIM_ConfigClock</b>	
Function Name	<code>_STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)</code>
Function Description	Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> <li>• <b>ClockFilter:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPTIM_CLK_FILTER_NONE</li> <li>- LL_LPTIM_CLK_FILTER_2</li> <li>- LL_LPTIM_CLK_FILTER_4</li> <li>- LL_LPTIM_CLK_FILTER_8</li> </ul> </li> <li>• <b>ClockPolarity:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPTIM_CLK_POLARITY_RISING</li> <li>- LL_LPTIM_CLK_POLARITY_FALLING</li> <li>- LL_LPTIM_CLK_POLARITY_RISING_FALLING</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> <li>• When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.</li> <li>• An internal clock source must be present when a digital filter is required for external clock.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR CKFLT LL_LPTIM_ConfigClock</li> <li>• CFGR CKPOL LL_LPTIM_ConfigClock</li> </ul>

**LL\_LPTIM\_GetClockPolarity**

Function Name	<code>_STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Get actual clock polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPTIM_CLK_POLARITY_RISING</li> <li>- LL_LPTIM_CLK_POLARITY_FALLING</li> <li>- LL_LPTIM_CLK_POLARITY_RISING_FALLING</li> </ul> </li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CFGR CKPOL LL\_LPTIM\_GetClockPolarity

### **LL\_LPTIM\_GetClockFilter**

Function Name      **`_STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter(LPTIM_TypeDef * LPTIMx)`**

Function Description      Get actual clock digital filter.

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **Returned:** value can be one of the following values:

- LL\_LPTIM\_CLK\_FILTER\_NONE
- LL\_LPTIM\_CLK\_FILTER\_2
- LL\_LPTIM\_CLK\_FILTER\_4
- LL\_LPTIM\_CLK\_FILTER\_8

- Reference Manual to  
LL API cross  
reference:
- CFGR CKFLT LL\_LPTIM\_GetClockFilter

### **LL\_LPTIM\_SetEncoderMode**

Function Name      **`_STATIC_INLINE void LL_LPTIM_SetEncoderMode(LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)`**

Function Description      Configure the encoder mode.

Parameters      • **LPTIMx:** Low-Power Timer instance

• **EncoderMode:** This parameter can be one of the following values:

- LL\_LPTIM\_ENCODER\_MODE\_RISING
- LL\_LPTIM\_ENCODER\_MODE\_FALLING
- LL\_LPTIM\_ENCODER\_MODE\_RISING\_FALLING

Return values      • **None:**

Notes      • This function must be called when the LPTIM instance is disabled.

- Reference Manual to  
LL API cross  
reference:
- CFGR CKPOL LL\_LPTIM\_SetEncoderMode

### **LL\_LPTIM\_GetEncoderMode**

Function Name      **`_STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode(LPTIM_TypeDef * LPTIMx)`**

Function Description      Get actual encoder mode.

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **Returned:** value can be one of the following values:

- LL\_LPTIM\_ENCODER\_MODE\_RISING
- LL\_LPTIM\_ENCODER\_MODE\_FALLING
- LL\_LPTIM\_ENCODER\_MODE\_RISING\_FALLING

Reference Manual to  
LL API cross  
reference:

- CFGR CKPOL LL\_LPTIM\_GetEncoderMode

### **LL\_LPTIM\_EnableEncoderMode**

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_EnableEncoderMode(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Enable the encoder mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> <li>• In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1.</li> <li>• LPTIM instance must be configured in continuous mode prior enabling the encoder mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR ENC LL_LPTIM_EnableEncoderMode</li> </ul>

### **LL\_LPTIM\_DisableEncoderMode**

Function Name	<b><code>_STATIC_INLINE void LL_LPTIM_DisableEncoderMode(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Disable the encoder mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function must be called when the LPTIM instance is disabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR ENC LL_LPTIM_DisableEncoderMode</li> </ul>

### **LL\_LPTIM\_IsEnabledEncoderMode**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode(LPTIM_TypeDef * LPTIMx)</code></b>
Function Description	Indicates whether the LPTIM operates in encoder mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR ENC LL_LPTIM_IsEnabledEncoderMode</li> </ul>

**LL\_LPTIM\_ClearFLAG\_CMPM**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_ClearFLAG_CMPM (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Clear the compare match flag (CMPMCF)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR CMPMCF LL_LPTIM_ClearFLAG_CMPM</li> </ul>

**LL\_LPTIM\_IsActiveFlag\_CMPM**

Function Name	<code>_STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPM (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Inform application whether a compare match interrupt has occurred.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR CMPM LL_LPTIM_IsActiveFlag_CMPM</li> </ul>

**LL\_LPTIM\_ClearFLAG\_ARRM**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_ClearFLAG_ARRM (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Clear the autoreload match flag (ARRMCF)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR ARRMCF LL_LPTIM_ClearFLAG_ARRM</li> </ul>

**LL\_LPTIM\_IsActiveFlag\_ARRM**

Function Name	<code>_STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Inform application whether a autoreload match interrupt has occurred.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ARRM LL_LPTIM_IsActiveFlag_ARRM</li> </ul>

**LL\_LPTIM\_ClearFlag\_EXTTRIG**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Clear the external trigger valid edge flag(EXTTRIGCF).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG</li> </ul>

**LL\_LPTIM\_IsActiveFlag\_EXTTRIG**

Function Name	<code>_STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Inform application whether a valid edge on the selected external trigger input has occurred.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG</li> </ul>

**LL\_LPTIM\_ClearFlag\_CMPOK**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_ClearFlag_CMPOK (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Clear the compare register update interrupt flag (CMPOKCF).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR CMPOKCF LL_LPTIM_ClearFlag_CMPOK</li> </ul>

**LL\_LPTIM\_IsActiveFlag\_CMPOK**

Function Name	<code>_STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPOK (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Informs application whether the APB bus write operation to the LPTIMx_CMP register has been successfully completed; If so, a new one can be initiated.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR CMPOK LL_LPTIM_IsActiveFlag_CMPOK</li> </ul>

**LL\_LPTIM\_ClearFlag\_ARROK**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Clear the autoreload register update interrupt flag (ARROKCF).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR ARROKCF LL_LPTIM_ClearFlag_ARROK</li> </ul>

**LL\_LPTIM\_IsActiveFlag\_ARROK**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed; If so, a new one can be initiated.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ARROK LL_LPTIM_IsActiveFlag_ARROK</li> </ul>

**LL\_LPTIM\_ClearFlag\_UP**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Clear the counter direction change to up interrupt flag (UPCF).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR UPCF LL_LPTIM_ClearFlag_UP</li> </ul>

**LL\_LPTIM\_IsActiveFlag\_UP**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (LPTIM_TypeDef * LPTIMx)</code>
Function Description	Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• ISR UP LL_LPTIM_IsActiveFlag_UP</li> </ul>

reference:

### **LL\_LPTIM\_ClearFlag\_DOWN**

Function Name      **\_\_STATIC\_INLINE void LL\_LPTIM\_ClearFlag\_DOWN(LPTIM\_TypeDef \* LPTIMx)**

Function Description      Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
reference:

- ICR DOWNCF LL\_LPTIM\_ClearFlag\_DOWN

### **LL\_LPTIM\_IsActiveFlag\_DOWN**

Function Name      **\_\_STATIC\_INLINE uint32\_t LL\_LPTIM\_IsActiveFlag\_DOWN(LPTIM\_TypeDef \* LPTIMx)**

Function Description      Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
reference:

- ISR DOWN LL\_LPTIM\_IsActiveFlag\_DOWN

### **LL\_LPTIM\_EnableIT\_CMPM**

Function Name      **\_\_STATIC\_INLINE void LL\_LPTIM\_EnableIT\_CMPM(LPTIM\_TypeDef \* LPTIMx)**

Function Description      Enable compare match interrupt (CMPMIE).

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
reference:

- IER CMPMIE LL\_LPTIM\_EnableIT\_CMPM

### **LL\_LPTIM\_DisableIT\_CMPM**

Function Name      **\_\_STATIC\_INLINE void LL\_LPTIM\_DisableIT\_CMPM(LPTIM\_TypeDef \* LPTIMx)**

Function Description      Disable compare match interrupt (CMPMIE).

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- IER CMPMIE LL\_LPTIM\_DisableIT\_CMPM

### **LL\_LPTIM\_IsEnabledIT\_CMPM**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPM(     LPTIM_TypeDef * LPTIMx)</code>
Function Description	Indicates whether the compare match interrupt (CMPMIE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IER CMPMIE LL_LPTIM_IsEnabledIT_CMPM</li> </ul>

### **LL\_LPTIM\_EnableIT\_ARRM**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM(     LPTIM_TypeDef * LPTIMx)</code>
Function Description	Enable autoreload match interrupt (ARRMIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IER ARRMIE LL_LPTIM_EnableIT_ARRM</li> </ul>

### **LL\_LPTIM\_DisableIT\_ARRM**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM(     LPTIM_TypeDef * LPTIMx)</code>
Function Description	Disable autoreload match interrupt (ARRMIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• IER ARRMIE LL_LPTIM_DisableIT_ARRM</li> </ul>

### **LL\_LPTIM\_IsEnabledIT\_ARRM**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM(     LPTIM_TypeDef * LPTIMx)</code>
Function Description	Indicates whether the autoreload match interrupt (ARRMIE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

- Reference Manual to LL API cross reference:
- IER ARRMIE LL\_LPTIM\_IsEnabledIT\_ARRM

### **LL\_LPTIM\_EnableIT\_EXTTRIG**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Enable external trigger valid edge interrupt (EXTTRIGIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to LL API cross reference:
- IER EXTTRIGIE LL\_LPTIM\_EnableIT\_EXTTRIG

### **LL\_LPTIM\_DisableIT\_EXTTRIG**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Disable external trigger valid edge interrupt (EXTTRIGIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to LL API cross reference:
- IER EXTTRIGIE LL\_LPTIM\_DisableIT\_EXTTRIG

### **LL\_LPTIM\_IsEnabledIT\_EXTTRIG**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

- Reference Manual to LL API cross reference:
- IER EXTTRIGIE LL\_LPTIM\_IsEnabledIT\_EXTTRIG

### **LL\_LPTIM\_EnableIT\_CMPOK**

Function Name	<code>__STATIC_INLINE void LL_LPTIM_EnableIT_CMPOK(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Enable compare register write completed interrupt (CMPOKIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to LL API cross reference:
- IER CMPOKIE LL\_LPTIM\_EnableIT\_CMPOK

### **LL\_LPTIM\_DisableIT\_CMPOK**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_DisableIT_CMPOK(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Disable compare register write completed interrupt (CMPOKIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to LL API cross reference:
- IER CMPOKIE LL\_LPTIM\_DisableIT\_CMPOK

### **LL\_LPTIM\_IsEnabledIT\_CMPOK**

Function Name	<code>_STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPOK(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Indicates whether the compare register write completed interrupt (CMPOKIE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

- Reference Manual to LL API cross reference:
- IER CMPOKIE LL\_LPTIM\_IsEnabledIT\_CMPOK

### **LL\_LPTIM\_EnableIT\_ARROK**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_EnableIT_ARROK(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Enable autoreload register write completed interrupt (ARROKIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to LL API cross reference:
- IER ARROKIE LL\_LPTIM\_EnableIT\_ARROK

### **LL\_LPTIM\_DisableIT\_ARROK**

Function Name	<code>_STATIC_INLINE void LL_LPTIM_DisableIT_ARROK(LPTIM_TypeDef * LPTIMx)</code>
Function Description	Disable autoreload register write completed interrupt (ARROKIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> Low-Power Timer instance</li> </ul>

- Return values
- **None:**

Reference Manual to  
LL API cross  
reference:

- IER ARROKIE LL\_LPTIM\_DisableIT\_ARROK

### **LL\_LPTIM\_IsEnabledIT\_ARROK**

Function Name      **`_STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK(LPTIM_TypeDef * LPTIMx)`**

Function Description      Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- IER ARROKIE LL\_LPTIM\_IsEnabledIT\_ARROK

### **LL\_LPTIM\_EnableIT\_UP**

Function Name      **`_STATIC_INLINE void LL_LPTIM_EnableIT_UP(LPTIM_TypeDef * LPTIMx)`**

Function Description      Enable direction change to up interrupt (UPIE).

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:

- IER UPIE LL\_LPTIM\_EnableIT\_UP

### **LL\_LPTIM\_DisableIT\_UP**

Function Name      **`_STATIC_INLINE void LL_LPTIM_DisableIT_UP(LPTIM_TypeDef * LPTIMx)`**

Function Description      Disable direction change to up interrupt (UPIE).

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:

- IER UPIE LL\_LPTIM\_DisableIT\_UP

### **LL\_LPTIM\_IsEnabledIT\_UP**

Function Name      **`_STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP(LPTIM_TypeDef * LPTIMx)`**

Function Description      Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters      • **LPTIMx:** Low-Power Timer instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- IER UPIE LL\_LPTIM\_IsEnabledIT\_UP

### **LL\_LPTIM\_EnableIT\_DOWN**

Function Name

**`_STATIC_INLINE void LL_LPTIM_EnableIT_DOWN(LPTIM_TypeDef * LPTIMx)`**

Function Description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- IER DOWNIE LL\_LPTIM\_EnableIT\_DOWN

### **LL\_LPTIM\_DisableIT\_DOWN**

Function Name

**`_STATIC_INLINE void LL_LPTIM_DisableIT_DOWN(LPTIM_TypeDef * LPTIMx)`**

Function Description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- IER DOWNIE LL\_LPTIM\_DisableIT\_DOWN

### **LL\_LPTIM\_IsEnabledIT\_DOWN**

Function Name

**`_STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN(LPTIM_TypeDef * LPTIMx)`**

Function Description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

- IER DOWNIE LL\_LPTIM\_IsEnabledIT\_DOWN

### **LL\_LPTIM\_DeInit**

Function Name

**`ErrorStatus LL_LPTIM_DeInit (LPTIM_TypeDef * LPTIMx)`**

Function Description

Set LPTIMx registers to their reset values.

Parameters

- **LPTIMx:** LP Timer instance

Return values

- **An:** ErrorStatus enumeration value:

- SUCCESS: LPTIMx registers are de-initialized
- ERROR: invalid LPTIMx instance

**LL\_LPTIM\_StructInit**

Function Name	<b>void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)</b>
Function Description	Set each fields of the LPTIM_InitStruct structure to its default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIM_InitStruct:</b> pointer to a LL_LPTIM_InitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_LPTIM\_Init**

Function Name	<b>ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, LL_LPTIM_InitTypeDef * LPTIM_InitStruct)</b>
Function Description	Configure the LPTIMx peripheral according to the specified parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> LP Timer Instance</li> <li>• <b>LPTIM_InitStruct:</b> pointer to a LL_LPTIM_InitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: LPTIMx instance has been initialized</li> <li>– ERROR: LPTIMx instance hasn't been initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• LL_LPTIM_Init can only be called when the LPTIM instance is disabled.</li> <li>• LPTIMx can be disabled using unitary function LL_LPTIM_Disable().</li> </ul>

## 66.3 LPTIM Firmware driver defines

### 66.3.1 LPTIM

#### *Clock Filter*

LL_LPTIM_CLK_FILTER_NONE	Any external clock signal level change is considered as a valid transition
LL_LPTIM_CLK_FILTER_2	External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition
LL_LPTIM_CLK_FILTER_4	External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition
LL_LPTIM_CLK_FILTER_8	External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

#### *Clock Polarity*

LL_LPTIM_CLK_POLARITY_RISING	The rising edge is the active edge used for counting
LL_LPTIM_CLK_POLARITY_FALLING	The falling edge is the active edge used

for counting

`LL_LPTIM_CLK_POLARITY_RISING_FALLING` Both edges are active edges

#### **Clock Source**

`LL_LPTIM_CLK_SOURCE_INTERNAL` LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

`LL_LPTIM_CLK_SOURCE_EXTERNAL` LPTIM is clocked by an external clock source through the LPTIM external Input1

#### **Counter Mode**

`LL_LPTIM_COUNTER_MODE_INTERNAL` The counter is incremented following each internal clock pulse

`LL_LPTIM_COUNTER_MODE_EXTERNAL` The counter is incremented following each valid clock pulse on the LPTIM external Input1

#### **Encoder Mode**

`LL_LPTIM_ENCODER_MODE_RISING` The rising edge is the active edge used for counting

`LL_LPTIM_ENCODER_MODE_FALLING` The falling edge is the active edge used for counting

`LL_LPTIM_ENCODER_MODE_RISING_FALLING` Both edges are active edges

#### **Get Flags Defines**

`LL_LPTIM_ISR_CMPM` Compare match

`LL_LPTIM_ISR_ARRM` Autoreload match

`LL_LPTIM_ISR_EXTRIG` External trigger edge event

`LL_LPTIM_ISR_CMPOK` Compare register update OK

`LL_LPTIM_ISR_ARROK` Autoreload register update OK

`LL_LPTIM_ISR_UP` Counter direction change down to up

`LL_LPTIM_ISR_DOWN` Counter direction change up to down

#### **IT Defines**

`LL_LPTIM_IER_CMPMIE` Compare match Interrupt Enable

`LL_LPTIM_IER_ARRMIE` Autoreload match Interrupt Enable

`LL_LPTIM_IER_EXTRIGIE` External trigger valid edge Interrupt Enable

`LL_LPTIM_IER_CMPOKIE` Compare register update OK Interrupt Enable

`LL_LPTIM_IER_ARROKIE` Autoreload register update OK Interrupt Enable

`LL_LPTIM_IER_UPIE` Direction change to UP Interrupt Enable

`LL_LPTIM_IER_DOWNIE` Direction change to down Interrupt Enable

#### **Operating Mode**

`LL_LPTIM_OPERATING_MODE_CONTINUOUS` LP Timer starts in continuous mode

`LL_LPTIM_OPERATING_MODE_ONESHOT` LP Tilmer starts in single mode

#### **Output Polarity**

<code>LL_LPTIM_OUTPUT_POLARITY_REGULAR</code>	The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CMP registers
<code>LL_LPTIM_OUTPUT_POLARITY_INVERSE</code>	The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CMP registers

***Output Waveform Type***

<code>LL_LPTIM_OUTPUT_WAVEFORM_PWM</code>	LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINUOUS or SINGLE
<code>LL_LPTIM_OUTPUT_WAVEFORM_SETONCE</code>	LPTIM generates a Set Once waveform

***Prescaler Value***

<code>LL_LPTIM_PRESCALER_DIV1</code>	Prescaler division factor is set to 1
<code>LL_LPTIM_PRESCALER_DIV2</code>	Prescaler division factor is set to 2
<code>LL_LPTIM_PRESCALER_DIV4</code>	Prescaler division factor is set to 4
<code>LL_LPTIM_PRESCALER_DIV8</code>	Prescaler division factor is set to 8
<code>LL_LPTIM_PRESCALER_DIV16</code>	Prescaler division factor is set to 16
<code>LL_LPTIM_PRESCALER_DIV32</code>	Prescaler division factor is set to 32
<code>LL_LPTIM_PRESCALER_DIV64</code>	Prescaler division factor is set to 64
<code>LL_LPTIM_PRESCALER_DIV128</code>	Prescaler division factor is set to 128

***Trigger Filter***

<code>LL_LPTIM_TRIG_FILTER_NONE</code>	Any trigger active level change is considered as a valid trigger
<code>LL_LPTIM_TRIG_FILTER_2</code>	Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger
<code>LL_LPTIM_TRIG_FILTER_4</code>	Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger
<code>LL_LPTIM_TRIG_FILTER_8</code>	Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

***Trigger Polarity***

<code>LL_LPTIM_TRIG_POLARITY_RISING</code>	LPTIM counter starts when a rising edge is detected
<code>LL_LPTIM_TRIG_POLARITY_FALLING</code>	LPTIM counter starts when a falling edge is detected
<code>LL_LPTIM_TRIG_POLARITY_RISING_FALLING</code>	LPTIM counter starts when a rising or a falling edge is detected

***Trigger Source***

<code>LL_LPTIM_TRIG_SOURCE_GPIO</code>	External input trigger is connected to TIMx_ETR input
<code>LL_LPTIM_TRIG_SOURCE_RTCALARMA</code>	External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB	External input trigger is connected to RTC Alarm B
LL_LPTIM_TRIG_SOURCE_RTCTAMP1	External input trigger is connected to RTC Tamper 1
LL_LPTIM_TRIG_SOURCE_RTCTAMP2	External input trigger is connected to RTC Tamper 2
LL_LPTIM_TRIG_SOURCE_RTCTAMP3	External input trigger is connected to RTC Tamper 3
LL_LPTIM_TRIG_SOURCE_COMP1	External input trigger is connected to COMP1 output
LL_LPTIM_TRIG_SOURCE_COMP2	External input trigger is connected to COMP2 output

***Update Mode***

LL_LPTIM_UPDATE_MODE_IMMEDIATE	Preload is disabled: registers are updated after each APB bus write access
LL_LPTIM_UPDATE_MODE_ENDOFFPERIOD	preload is enabled: registers are updated at the end of the current LPTIM period

***Common Write and read registers Macros*****LL\_LPTIM\_WriteReg    Description:**

- Write a value in LPTIM register.

**Parameters:**

- \_\_INSTANCE\_\_: LPTIM Instance
- \_\_REG\_\_: Register to be written
- \_\_VALUE\_\_: Value to be written in the register

**Return value:**

- None

**LL\_LPTIM\_ReadReg****Description:**

- Read a value in LPTIM register.

**Parameters:**

- \_\_INSTANCE\_\_: LPTIM Instance
- \_\_REG\_\_: Register to be read

**Return value:**

- Register: value

## 67 LL LPUART Generic Driver

### 67.1 LPUART Firmware driver registers structures

#### 67.1.1 LL\_LPUART\_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\_LPUART\_InitTypeDef::BaudRate***  
This field defines expected LPUART communication baud rate. This feature can be modified afterwards using unitary function **LL\_LPUART\_SetBaudRate()**.
- ***uint32\_t LL\_LPUART\_InitTypeDef::DataWidth***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of **LPUART\_LL\_EC\_DATAWIDTH**. This feature can be modified afterwards using unitary function **LL\_LPUART\_SetDataWidth()**.
- ***uint32\_t LL\_LPUART\_InitTypeDef::StopBits***  
Specifies the number of stop bits transmitted. This parameter can be a value of **LPUART\_LL\_EC\_STOPBITS**. This feature can be modified afterwards using unitary function **LL\_LPUART\_SetStopBitsLength()**.
- ***uint32\_t LL\_LPUART\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of **LPUART\_LL\_EC\_PARITY**. This feature can be modified afterwards using unitary function **LL\_LPUART\_SetParity()**.
- ***uint32\_t LL\_LPUART\_InitTypeDef::TransferDirection***  
Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of **LPUART\_LL\_EC\_DIRECTION**. This feature can be modified afterwards using unitary function **LL\_LPUART\_SetTransferDirection()**.
- ***uint32\_t LL\_LPUART\_InitTypeDef::HardwareFlowControl***  
Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of **LPUART\_LL\_EC\_HWCONTROL**. This feature can be modified afterwards using unitary function **LL\_LPUART\_SetHWFlowCtrl()**.

### 67.2 LPUART Firmware driver API description

#### 67.2.1 Detailed description of functions

##### LL\_LPUART\_Enable

Function Name      **\_STATIC\_INLINE void LL\_LPUART\_Enable (USART\_TypeDef**

**\* LPUARTx)**

Function Description	LPUART Enable.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UE LL_LPUART_Enable</li> </ul>

**LL\_LPUART\_Disable**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_Disable( (USART_TypeDef * LPUARTx)</code></b>
Function Description	LPUART Disable.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.</li> <li>• In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UE LL_LPUART_Disable</li> </ul>

**LL\_LPUART\_IsEnabled**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsEnabled( (USART_TypeDef * LPUARTx)</code></b>
Function Description	Indicate if LPUART is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UE LL_LPUART_IsEnabled</li> </ul>

**LL\_LPUART\_EnableInStopMode**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_EnableInStopMode( (USART_TypeDef * LPUARTx)</code></b>
Function Description	LPUART enabled in STOP Mode.

Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 UESM LL_LPUART_EnableInStopMode</li> </ul>

### LL\_LPUART\_DisableInStopMode

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableInStopMode(USART_TypeDef * LPUARTx)</code></b>
Function Description	LPUART disabled in STOP Mode.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When this function is disabled, LPUART is not able to wake up the MCU from Stop mode</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 UESM LL_LPUART_DisableInStopMode</li> </ul>

### LL\_LPUART\_IsEnabledInStopMode

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode(USART_TypeDef * LPUARTx)</code></b>
Function Description	Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 UESM LL_LPUART_IsEnabledInStopMode</li> </ul>

### LL\_LPUART\_EnableDirectionRx

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_EnableDirectionRx(USART_TypeDef * LPUARTx)</code></b>
Function Description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 RE LL_LPUART_EnableDirectionRx</li> </ul>

**LL\_LPUART\_DisableDirectionRx**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)</code></b>
Function Description	Receiver Disable.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 RE LL_LPUART_DisableDirectionRx</li> </ul>

**LL\_LPUART\_EnableDirectionTx**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)</code></b>
Function Description	Transmitter Enable.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TE LL_LPUART_EnableDirectionTx</li> </ul>

**LL\_LPUART\_DisableDirectionTx**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)</code></b>
Function Description	Transmitter Disable.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TE LL_LPUART_DisableDirectionTx</li> </ul>

**LL\_LPUART\_SetTransferDirection**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)</code></b>
Function Description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>TransferDirection:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_LPUART_DIRECTION_NONE</code></li> <li>- <code>LL_LPUART_DIRECTION_RX</code></li> <li>- <code>LL_LPUART_DIRECTION_TX</code></li> <li>- <code>LL_LPUART_DIRECTION_TX_RX</code></li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 RE LL_LPUART_SetTransferDirection</li> <li>CR1 TE LL_LPUART_SetTransferDirection</li> </ul>

### LL\_LPUART\_GetTransferDirection

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection(     USART_TypeDef * LPUARTx)</code>
Function Description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>LL_LPUART_DIRECTION_NONE</li> <li>LL_LPUART_DIRECTION_RX</li> <li>LL_LPUART_DIRECTION_TX</li> <li>LL_LPUART_DIRECTION_TX_RX</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 RE LL_LPUART_GetTransferDirection</li> <li>CR1 TE LL_LPUART_GetTransferDirection</li> </ul>

### LL\_LPUART\_SetParity

Function Name	<code>__STATIC_INLINE void LL_LPUART_SetParity(     USART_TypeDef * LPUARTx, uint32_t Parity)</code>
Function Description	Configure Parity (enabled/disabled and parity mode if enabled)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> <li><b>Parity:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>LL_LPUART_PARITY_NONE</li> <li>LL_LPUART_PARITY_EVEN</li> <li>LL_LPUART_PARITY_ODD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>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 (depending on data width) and parity is checked on the received data.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 PS LL_LPUART_SetParity</li> <li>CR1 PCE LL_LPUART_SetParity</li> </ul>

### LL\_LPUART\_GetParity

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_GetParity(     USART_TypeDef * LPUARTx)</code>
Function Description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_LPUART_PARITY_NONE</li> <li>– LL_LPUART_PARITY_EVEN</li> <li>– LL_LPUART_PARITY_ODD</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PS LL_LPUART_GetParity</li> <li>• CR1 PCE LL_LPUART_GetParity</li> </ul>

### LL\_LPUART\_SetWakeUpMethod

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetWakeUpMethod( USART_TypeDef * LPUARTx, uint32_t Method)</code></b>
Function Description	Set Receiver Wake Up method from Mute mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>Method:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_LPUART_WAKEUP_IDLELINE</li> <li>– LL_LPUART_WAKEUP_ADDRESSMARK</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 WAKE LL_LPUART_SetWakeUpMethod</li> </ul>

### LL\_LPUART\_GetWakeUpMethod

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod( USART_TypeDef * LPUARTx)</code></b>
Function Description	Return Receiver Wake Up method from Mute mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_LPUART_WAKEUP_IDLELINE</li> <li>– LL_LPUART_WAKEUP_ADDRESSMARK</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 WAKE LL_LPUART_GetWakeUpMethod</li> </ul>

### LL\_LPUART\_SetDataWidth

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetDataWidth( USART_TypeDef * LPUARTx, uint32_t DataWidth)</code></b>
Function Description	Set Word length (nb of data bits, excluding start and stop bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>DataWidth:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_LPUART_DATAWIDTH_7B</li> <li>– LL_LPUART_DATAWIDTH_8B</li> <li>– LL_LPUART_DATAWIDTH_9B</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- CR1 M LL\_LPUART\_SetDataWidth  
LL API cross reference:

### **LL\_LPUART\_GetDataWidth**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth( USART_TypeDef * LPUARTx)</code></b>
Function Description	Return Word length (i.e.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPUART_DATAWIDTH_7B</li> <li>- LL_LPUART_DATAWIDTH_8B</li> <li>- LL_LPUART_DATAWIDTH_9B</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 M LL_LPUART_SetDataWidth</li> </ul>

### **LL\_LPUART\_EnableMuteMode**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_EnableMuteMode( USART_TypeDef * LPUARTx)</code></b>
Function Description	Allow switch between Mute Mode and Active mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 MME LL_LPUART_EnableMuteMode</li> </ul>

### **LL\_LPUART\_DisableMuteMode**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableMuteMode( USART_TypeDef * LPUARTx)</code></b>
Function Description	Prevent Mute Mode use.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 MME LL_LPUART_DisableMuteMode</li> </ul>

### **LL\_LPUART\_IsEnabledMuteMode**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode( USART_TypeDef * LPUARTx)</code></b>
Function Description	Indicate if switch between Mute Mode and Active mode is allowed.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>

- |   |   |
|---|---|
| Return values                                     | <ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>      |
| Reference Manual to<br>LL API cross<br>reference: | <ul style="list-style-type: none"> <li>• CR1 MME LL_LPUART_IsEnabledMuteMode</li> </ul> |

### **LL\_LPUART\_SetStopBitsLength**

- |   |  |
|---|--|
| Function Name                                     | <b><code>_STATIC_INLINE void LL_LPUART_SetStopBitsLength(<br/>(USART_TypeDef * LPUARTx, uint32_t StopBits)</code></b>  |
| Function Description                              | Set the length of the stop bits.   |
| Parameters  | <ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>StopBits:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPUART_STOPBITS_1</li> <li>- LL_LPUART_STOPBITS_2</li> </ul> </li> </ul> |
| Return values                                     | <ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>   |
| Reference Manual to<br>LL API cross<br>reference: | <ul style="list-style-type: none"> <li>• CR2 STOP LL_LPUART_SetStopBitsLength</li> </ul>   |

### **LL\_LPUART\_GetStopBitsLength**

- |   |  |
|---|--|
| Function Name                                     | <b><code>_STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength(<br/>(USART_TypeDef * LPUARTx)</code></b>   |
| Function Description                              | Retrieve the length of the stop bits.  |
| Parameters  | <ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>  |
| Return values                                     | <ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPUART_STOPBITS_1</li> <li>- LL_LPUART_STOPBITS_2</li> </ul> </li> </ul> |
| Reference Manual to<br>LL API cross<br>reference: | <ul style="list-style-type: none"> <li>• CR2 STOP LL_LPUART_GetStopBitsLength</li> </ul>   |

### **LL\_LPUART\_ConfigCharacter**

- |                      |  |
|----------------------|--|
| Function Name        | <b><code>_STATIC_INLINE void LL_LPUART_ConfigCharacter(<br/>(USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t<br/>Parity, uint32_t StopBits)</code></b>   |
| Function Description | Configure Character frame format (Datawidth, Parity control, Stop Bits)  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>DataWidth:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPUART_DATAWIDTH_7B</li> <li>- LL_LPUART_DATAWIDTH_8B</li> <li>- LL_LPUART_DATAWIDTH_9B</li> </ul> </li> <li>• <b>Parity:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPUART_PARITY_NONE</li> <li>- LL_LPUART_PARITY_EVEN</li> </ul> </li> </ul> |

Return values	<ul style="list-style-type: none"> <li>- LL_LPUART_PARITY_ODD</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• <b>StopBits:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_STOPBITS_1</li> <li>- LL_LPUART_STOPBITS_2</li> </ul> </li> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function</li> <li>• CR1 PS LL_LPUART_ConfigCharacter</li> <li>• CR1 PCE LL_LPUART_ConfigCharacter</li> <li>• CR1 M LL_LPUART_ConfigCharacter</li> <li>• CR2 STOP LL_LPUART_ConfigCharacter</li> </ul>

### LL\_LPUART\_SetTXRXSwap

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_SetTXRXSwap( (USART_TypeDef * LPUARTx, uint32_t SwapConfig)</code></b>
Function Description	Configure TX/RX pins swapping setting.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>SwapConfig:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_TXRX_STANDARD</li> <li>- LL_LPUART_TXRX_SWAPPED</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 SWAP LL_LPUART_SetTXRXSwap</li> </ul>

### LL\_LPUART\_GetTXRXSwap

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap( (USART_TypeDef * LPUARTx)</code></b>
Function Description	Retrieve TX/RX pins swapping configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_TXRX_STANDARD</li> <li>- LL_LPUART_TXRX_SWAPPED</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 SWAP LL_LPUART_GetTXRXSwap</li> </ul>

### LL\_LPUART\_SetRXPinLevel

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_SetRXPinLevel( (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)</code></b>
---------------	---

Function Description	Configure RX pin active level logic.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>PinInvMethod:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPUART_RXPIN_LEVEL_STANDARD</li> <li>– LL_LPUART_RXPIN_LEVEL_INVERTED</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RXINV LL_LPUART_SetRXPinLevel</li> </ul>

### LL\_LPUART\_GetRXPinLevel

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel(     USART_TypeDef * LPUARTx)</code>
Function Description	Retrieve RX pin active level logic configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPUART_RXPIN_LEVEL_STANDARD</li> <li>– LL_LPUART_RXPIN_LEVEL_INVERTED</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RXINV LL_LPUART_GetRXPinLevel</li> </ul>

### LL\_LPUART\_SetTXPinLevel

Function Name	<code>__STATIC_INLINE void LL_LPUART_SetTXPinLevel(     USART_TypeDef * LPUARTx, uint32_t PinInvMethod)</code>
Function Description	Configure TX pin active level logic.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>PinInvMethod:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPUART_TXPIN_LEVEL_STANDARD</li> <li>– LL_LPUART_TXPIN_LEVEL_INVERTED</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TXINV LL_LPUART_SetTXPinLevel</li> </ul>

### LL\_LPUART\_GetTXPinLevel

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel(     USART_TypeDef * LPUARTx)</code>
Function Description	Retrieve TX pin active level logic configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_LPUART_TXPIN_LEVEL_STANDARD</li> </ul> </li> </ul>

- LL\_LPUART\_TXPIN\_LEVEL\_INVERTED
  - CR2 TXINV LL\_LPUART\_GetTXPinLevel
- Reference Manual to  
LL API cross  
reference:

### **LL\_LPUART\_SetBinaryDataLogic**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic( USART_TypeDef * LPUARTx, uint32_t DataLogic)</code></b>
Function Description	Configure Binary data logic.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>DataLogic:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_BINARY_LOGIC_POSITIVE</li> <li>- LL_LPUART_BINARY_LOGIC_NEGATIVE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 DATAINV LL_LPUART_SetBinaryDataLogic</li> </ul>

### **LL\_LPUART\_GetBinaryDataLogic**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic( USART_TypeDef * LPUARTx)</code></b>
Function Description	Retrieve Binary data configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_BINARY_LOGIC_POSITIVE</li> <li>- LL_LPUART_BINARY_LOGIC_NEGATIVE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 DATAINV LL_LPUART_SetBinaryDataLogic</li> </ul>

### **LL\_LPUART\_SetTransferBitOrder**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetTransferBitOrder( USART_TypeDef * LPUARTx, uint32_t BitOrder)</code></b>
Function Description	Configure transfer bit order (either Less or Most Significant Bit First)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>BitOrder:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_BITORDER_LSBFIRST</li> <li>- LL_LPUART_BITORDER_MSBFIRST</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

---

Notes	<ul style="list-style-type: none"> <li>MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 MSBFIRST LL_LPUART_SetTransferBitOrder</li> </ul>

### LL\_LPUART\_GetTransferBitOrder

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder(     USART_TypeDef * LPUARTx)</code>
Function Description	Return transfer bit order (either Less or Most Significant Bit First)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_LPUART_BITORDER_LSBFIRST</li> <li>- LL_LPUART_BITORDER_MSBFIRST</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 MSBFIRST LL_LPUART_GetTransferBitOrder</li> </ul>

### LL\_LPUART\_ConfigNodeAddress

Function Name	<code>__STATIC_INLINE void LL_LPUART_ConfigNodeAddress(     USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t     NodeAddress)</code>
Function Description	Set Address of the LPUART node.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> <li><b>AddressLen:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_LPUART_ADDRESS_DETECT_4B</li> <li>- LL_LPUART_ADDRESS_DETECT_7B</li> </ul> </li> <li><b>NodeAddress:</b> 4 or 7 bit Address of the LPUART node.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.</li> <li>4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is</li> </ul>

Reference Manual to  
LL API cross  
reference:

compared to the ADD[7:0] value and CMF flag is set on  
match)

- CR2 ADD LL\_LPUART\_ConfigNodeAddress
- CR2 ADDM7 LL\_LPUART\_ConfigNodeAddress

### **LL\_LPUART\_GetNodeAddress**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress( (USART_TypeDef * LPUARTx)</code></b>
Function Description	Return 8 bit Address of the LPUART node as set in ADD field of CR2.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Address:</b> of the LPUART node (Value between Min_Data=0 and Max_Data=255)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ADD LL_LPUART_GetNodeAddress</li> </ul>

### **LL\_LPUART\_GetNodeAddressLen**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen( (USART_TypeDef * LPUARTx)</code></b>
Function Description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_LPUART_ADDRESS_DETECT_4B</li> <li>- LL_LPUART_ADDRESS_DETECT_7B</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ADDM7 LL_LPUART_GetNodeAddressLen</li> </ul>

### **LL\_LPUART\_EnableRTSHWFlowCtrl**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl( (USART_TypeDef * LPUARTx)</code></b>
Function Description	Enable RTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl</li> </ul>

reference:

### **LL\_LPUART\_DisableRTSHWFlowCtrl**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable RTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl</li> </ul>

### **LL\_LPUART\_EnableCTSHWFlowCtrl**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Enable CTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl</li> </ul>

### **LL\_LPUART\_DisableCTSHWFlowCtrl**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable CTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl</li> </ul>

### **LL\_LPUART\_SetHWFlowCtrl**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl(     USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)</code></b>
Function Description	Configure HW Flow Control mode (both CTS and RTS)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>HardwareFlowControl:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>LL_LPUART_HWCONTROL_NONE</code></li> <li>– <code>LL_LPUART_HWCONTROL_RTS</code></li> <li>– <code>LL_LPUART_HWCONTROL_CTS</code></li> </ul> </li> </ul>

– LL\_LPUART\_HWCONTROL\_RTS\_CTS

**Return values**

- **None:**
- CR3 RTSE LL\_LPUART\_SetHWFlowCtrl
- CR3 CTSE LL\_LPUART\_SetHWFlowCtrl

Reference Manual to  
LL API cross  
reference:

### LL\_LPUART\_GetHWFlowCtrl

Function Name **`_STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl(USART_TypeDef * LPUARTx)`**

Function Description Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_LPUART\_HWCONTROL\_NONE
  - LL\_LPUART\_HWCONTROL\_RTS
  - LL\_LPUART\_HWCONTROL\_CTS
  - LL\_LPUART\_HWCONTROL\_RTS\_CTS

Reference Manual to  
LL API cross  
reference:

- CR3 RTSE LL\_LPUART\_GetHWFlowCtrl
- CR3 CTSE LL\_LPUART\_GetHWFlowCtrl

### LL\_LPUART\_EnableOverrunDetect

Function Name **`_STATIC_INLINE void LL_LPUART_EnableOverrunDetect(USART_TypeDef * LPUARTx)`**

Function Description Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

**Return values**

- **None:**

Reference Manual to  
LL API cross  
reference:

- CR3 OVRDIS LL\_LPUART\_EnableOverrunDetect

### LL\_LPUART\_DisableOverrunDetect

Function Name **`_STATIC_INLINE void LL_LPUART_DisableOverrunDetect(USART_TypeDef * LPUARTx)`**

Function Description Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

**Return values**

- **None:**

Reference Manual to  
LL API cross  
reference:

- CR3 OVRDIS LL\_LPUART\_DisableOverrunDetect

### LL\_LPUART\_IsEnabledOverrunDetect

Function Name **`_STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect(USART_TypeDef *`**

**LPUARTx)**

Function Description	Indicate if Overrun detection is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

**LL\_LPUART\_SetWKUPType**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)</code></b>
Function Description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>Type:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_WAKEUP_ON_ADDRESS</li> <li>- LL_LPUART_WAKEUP_ON_STARTBIT</li> <li>- LL_LPUART_WAKEUP_ON_RXNE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	CR3 WUS LL_LPUART_SetWKUPType

**LL\_LPUART\_GetWKUPType**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (USART_TypeDef * LPUARTx)</code></b>
Function Description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_LPUART_WAKEUP_ON_ADDRESS</li> <li>- LL_LPUART_WAKEUP_ON_STARTBIT</li> <li>- LL_LPUART_WAKEUP_ON_RXNE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	CR3 WUS LL_LPUART_GetWKUPType

**LL\_LPUART\_SetBaudRate**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t BaudRate)</code></b>
Function Description	Configure LPUART BRR register for achieving expected Baud Rate value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>PeriphClk:</b> Peripheral Clock</li> </ul>

- **BaudRate:** Baud Rate
- **None:**
- Return values
- Notes
  - Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
  - Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value != 0).
  - Provided that LPUARTx\_BRR must be >= 0x300 and LPUART\_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].
- Reference Manual to LL API cross reference:
  - BRR BRR LL\_LPUART\_SetBaudRate

### **LL\_LPUART\_GetBaudRate**

- |   |   |
|---|---|
| Function Name                               | <b><code>__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate(<br/>USART_TypeDef * LPUARTx, uint32_t PeriphClk)</code></b>                              |
| Function Description                        | Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.           |
| Parameters                                  | <ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>PeriphClk:</b> Peripheral Clock</li> </ul>                 |
| Return values                               | <ul style="list-style-type: none"> <li>• <b>Baud:</b> Rate</li> </ul>   |
| Notes                                       | <ul style="list-style-type: none"> <li>• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.</li> </ul> |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none"> <li>• BRR BRR LL_LPUART_GetBaudRate</li> </ul>   |

### **LL\_LPUART\_EnableHalfDuplex**

- |   |   |
|---|---|
| Function Name                               | <b><code>__STATIC_INLINE void LL_LPUART_EnableHalfDuplex(<br/>USART_TypeDef * LPUARTx)</code></b> |
| Function Description                        | Enable Single Wire Half-Duplex mode.  |
| Parameters                                  | <ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>               |
| Return values                               | <ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>                                  |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none"> <li>• CR3 HDSEL LL_LPUART_EnableHalfDuplex</li> </ul>          |

### **LL\_LPUART\_DisableHalfDuplex**

- |               |  |
|---------------|--|
| Function Name | <b><code>__STATIC_INLINE void LL_LPUART_DisableHalfDuplex(<br/>USART_TypeDef * LPUARTx)</code></b> |
|---------------|--|

Function Description	Disable Single Wire Half-Duplex mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 HDSEL LL_LPUART_DisableHalfDuplex</li> </ul>

### **LL\_LPUART\_IsEnabledHalfDuplex**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex( USART_TypeDef * LPUARTx)</code></b>
Function Description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex</li> </ul>

### **LL\_LPUART\_SetDEDeassertionTime**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime( USART_TypeDef * LPUARTx, uint32_t Time)</code></b>
Function Description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>Time:</b> Value between Min_Data=0 and Max_Data=31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DEDT LL_LPUART_SetDEDeassertionTime</li> </ul>

### **LL\_LPUART\_GetDEDeassertionTime**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime(USART_TypeDef * LPUARTx)</code></b>
Function Description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Time:</b> value expressed on 5 bits ([4:0] bits) : c</li> </ul>

Reference Manual to  
LL API cross  
reference:

- CR1 DEDT LL\_LPUART\_GetDEDeassertionTime

### **LL\_LPUART\_SetDEAssertionTime**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_SetDEAssertionTime</code></b>
---------------	---

**(USART\_TypeDef \* LPUARTx, uint32\_t Time)**

Function Description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>Time:</b> Value between Min_Data=0 and Max_Data=31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DEAT LL_LPUART_SetDEAssertionTime</li> </ul>

**LL\_LPUART\_GetDEAssertionTime**

Function Name	<b>STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (USART_TypeDef * LPUARTx)</b>
Function Description	Return DEAT (Driver Enable Assertion Time)
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Time:</b> value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DEAT LL_LPUART_GetDEAssertionTime</li> </ul>

**LL\_LPUART\_EnableDEMode**

Function Name	<b>STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)</b>
Function Description	Enable Driver Enable (DE) Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 DEM LL_LPUART_EnableDEMode</li> </ul>

**LL\_LPUART\_DisableDEMode**

Function Name	<b>STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)</b>
Function Description	Disable Driver Enable (DE) Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 DEM LL_LPUART_DisableDEMode</li> </ul>

**LL\_LPUART\_IsEnabledDEMode**

Function Name      **`_STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode  
(USART_TypeDef * LPUARTx)`**

Function Description      Indicate if Driver Enable (DE) Mode is enabled.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
• CR3 DEM LL\_LPUART\_IsEnabledDEMode

**LL\_LPUART\_SetDESignalPolarity**

Function Name      **`_STATIC_INLINE void LL_LPUART_SetDESignalPolarity  
(USART_TypeDef * LPUARTx, uint32_t Polarity)`**

Function Description      Select Driver Enable Polarity.

Parameters      • **LPUARTx:** LPUART Instance

• **Polarity:** This parameter can be one of the following values:  
– LL\_LPUART\_DE\_POLARITY\_HIGH  
– LL\_LPUART\_DE\_POLARITY\_LOW

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
• CR3 DEP LL\_LPUART\_SetDESignalPolarity

**LL\_LPUART\_GetDESignalPolarity**

Function Name      **`_STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity  
(USART_TypeDef * LPUARTx)`**

Function Description      Return Driver Enable Polarity.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **Returned:** value can be one of the following values:  
– LL\_LPUART\_DE\_POLARITY\_HIGH  
– LL\_LPUART\_DE\_POLARITY\_LOW

Reference Manual to  
LL API cross  
reference:  
• CR3 DEP LL\_LPUART\_GetDESignalPolarity

**LL\_LPUART\_IsActiveFlag\_PE**

Function Name      **`_STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE  
(USART_TypeDef * LPUARTx)`**

Function Description      Check if the LPUART Parity Error Flag is set or not.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
ISR PE LL\_LPUART\_IsActiveFlag\_PE

LL API cross  
reference:

### **LL\_LPUART\_IsActiveFlag\_FE**

Function Name      **STATIC\_INLINE uint32\_t LL\_LPUART\_IsActiveFlag\_FE  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART Framing Error Flag is set or not.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

### **LL\_LPUART\_IsActiveFlag\_NE**

Function Name      **STATIC\_INLINE uint32\_t LL\_LPUART\_IsActiveFlag\_NE  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART Noise detected Flag is set or not.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

### **LL\_LPUART\_IsActiveFlag\_ORE**

Function Name      **STATIC\_INLINE uint32\_t LL\_LPUART\_IsActiveFlag\_ORE  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART OverRun Error Flag is set or not.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:

### **LL\_LPUART\_IsActiveFlag\_IDLE**

Function Name      **STATIC\_INLINE uint32\_t LL\_LPUART\_IsActiveFlag\_IDLE  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART IDLE line detected Flag is set or not.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross

reference:

### **LL\_LPUART\_IsActiveFlag\_RXNE**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Read Data Register Not Empty Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR RXNE LL_LPUART_IsActiveFlag_RXNE</li> </ul>

### **LL\_LPUART\_IsActiveFlag\_TC**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Transmission Complete Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TC LL_LPUART_IsActiveFlag_TC</li> </ul>

### **LL\_LPUART\_IsActiveFlag\_TXE**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Transmit Data Register Empty Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TXE LL_LPUART_IsActiveFlag_TXE</li> </ul>

### **LL\_LPUART\_IsActiveFlag\_nCTS**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART CTS interrupt Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• ISR CTSIF LL_LPUART_IsActiveFlag_nCTS</li> </ul>

reference:

### **LL\_LPUART\_IsActiveFlag\_CTS**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART CTS Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR CTS LL_LPUART_IsActiveFlag_CTS</li> </ul>

### **LL\_LPUART\_IsActiveFlag\_BUSY**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Busy Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR BUSY LL_LPUART_IsActiveFlag_BUSY</li> </ul>

### **LL\_LPUART\_IsActiveFlag\_CM**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Character Match Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR CMF LL_LPUART_IsActiveFlag_CM</li> </ul>

### **LL\_LPUART\_IsActiveFlag\_SBK**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK(     USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Send Break Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR SBKF LL_LPUART_IsActiveFlag_SBK</li> </ul>

**LL\_LPUART\_IsActiveFlag\_RWU**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU(   USART_TypeDef * LPUARTx)</code>
Function Description	Check if the LPUART Receive Wake Up from mute mode Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR RWU LL_LPUART_IsActiveFlag_RWU</li> </ul>

**LL\_LPUART\_IsActiveFlag\_WKUP**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP(   USART_TypeDef * LPUARTx)</code>
Function Description	Check if the LPUART Wake Up from stop mode Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR WUF LL_LPUART_IsActiveFlag_WKUP</li> </ul>

**LL\_LPUART\_IsActiveFlag\_TEACK**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK(   USART_TypeDef * LPUARTx)</code>
Function Description	Check if the LPUART Transmit Enable Acknowledge Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TEACK LL_LPUART_IsActiveFlag_TEACK</li> </ul>

**LL\_LPUART\_IsActiveFlag\_REACK**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK(   USART_TypeDef * LPUARTx)</code>
Function Description	Check if the LPUART Receive Enable Acknowledge Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR REACK LL_LPUART_IsActiveFlag_REACK</li> </ul>

**LL\_LPUART\_ClearFlag\_PE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)</code>
Function Description	Clear Parity Error Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR PECF LL_LPUART_ClearFlag_PE</li> </ul>

**LL\_LPUART\_ClearFlag\_FE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)</code>
Function Description	Clear Framing Error Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR FECF LL_LPUART_ClearFlag_FE</li> </ul>

**LL\_LPUART\_ClearFlag\_NE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)</code>
Function Description	Clear Noise detected Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR NCF LL_LPUART_ClearFlag_NE</li> </ul>

**LL\_LPUART\_ClearFlag\_ORE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)</code>
Function Description	Clear OverRun Error Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR ORECF LL_LPUART_ClearFlag_ORE</li> </ul>

**LL\_LPUART\_ClearFlag\_IDLE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE(     USART_TypeDef * LPUARTx)</code>
Function Description	Clear IDLE line detected Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR IDLECF LL_LPUART_ClearFlag_IDLE</li> </ul>

**LL\_LPUART\_ClearFlag\_TC**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_TC(     USART_TypeDef * LPUARTx)</code>
Function Description	Clear Transmission Complete Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR TCCF LL_LPUART_ClearFlag_TC</li> </ul>

**LL\_LPUART\_ClearFlag\_nCTS**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS(     USART_TypeDef * LPUARTx)</code>
Function Description	Clear CTS Interrupt Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR CTSCF LL_LPUART_ClearFlag_nCTS</li> </ul>

**LL\_LPUART\_ClearFlag\_CM**

Function Name	<code>__STATIC_INLINE void LL_LPUART_ClearFlag_CM(     USART_TypeDef * LPUARTx)</code>
Function Description	Clear Character Match Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR CMCF LL_LPUART_ClearFlag_CM</li> </ul>

**LL\_LPUART\_ClearFlag\_WKUP**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)</code></b>
Function Description	Clear Wake Up from stop mode Flag.
Parameters	<ul style="list-style-type: none"><li>• <b>LPUARTx:</b> LPUART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ICR WUCF LL_LPUART_ClearFlag_WKUP</li></ul>

**LL\_LPUART\_EnableIT\_IDLE**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)</code></b>
Function Description	Enable IDLE Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>LPUARTx:</b> LPUART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 IDLEIE LL_LPUART_EnableIT_IDLE</li></ul>

**LL\_LPUART\_EnableIT\_RXNE**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_EnableIT_RXNE (USART_TypeDef * LPUARTx)</code></b>
Function Description	Enable RX Not Empty Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>LPUARTx:</b> LPUART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 RXNEIE LL_LPUART_EnableIT_RXNE</li></ul>

**LL\_LPUART\_EnableIT\_TC**

Function Name	<b><code>_STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)</code></b>
Function Description	Enable Transmission Complete Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>LPUARTx:</b> LPUART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 TCIE LL_LPUART_EnableIT_TC</li></ul>

**LL\_LPUART\_EnableIT\_TXE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_EnableIT_TXE(     USART_TypeDef * LPUARTx)</code>
Function Description	Enable TX Empty Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TXEIE LL_LPUART_EnableIT_TXE</li> </ul>

**LL\_LPUART\_EnableIT\_PE**

Function Name	<code>__STATIC_INLINE void LL_LPUART_EnableIT_PE(     USART_TypeDef * LPUARTx)</code>
Function Description	Enable Parity Error Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PEIE LL_LPUART_EnableIT_PE</li> </ul>

**LL\_LPUART\_EnableIT\_CM**

Function Name	<code>__STATIC_INLINE void LL_LPUART_EnableIT_CM(     USART_TypeDef * LPUARTx)</code>
Function Description	Enable Character Match Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CMIE LL_LPUART_EnableIT_CM</li> </ul>

**LL\_LPUART\_EnableIT\_ERROR**

Function Name	<code>__STATIC_INLINE void LL_LPUART_EnableIT_ERROR(     USART_TypeDef * LPUARTx)</code>
Function Description	Enable Error Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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 LPUARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.</li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CR3 EIE LL\_LPUART\_EnableIT\_ERROR

### **LL\_LPUART\_EnableIT\_CTS**

Function Name      **\_\_STATIC\_INLINE void LL\_LPUART\_EnableIT\_CTS  
(USART\_TypeDef \* LPUARTx)**

Function Description      Enable CTS Interrupt.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR3 CTSIE LL\_LPUART\_EnableIT\_CTS

### **LL\_LPUART\_EnableIT\_WKUP**

Function Name      **\_\_STATIC\_INLINE void LL\_LPUART\_EnableIT\_WKUP  
(USART\_TypeDef \* LPUARTx)**

Function Description      Enable Wake Up from Stop Mode Interrupt.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR3 WUFIE LL\_LPUART\_EnableIT\_WKUP

### **LL\_LPUART\_DisableIT\_IDLE**

Function Name      **\_\_STATIC\_INLINE void LL\_LPUART\_DisableIT\_IDLE  
(USART\_TypeDef \* LPUARTx)**

Function Description      Disable IDLE Interrupt.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR1 IDLEIE LL\_LPUART\_DisableIT\_IDLE

### **LL\_LPUART\_DisableIT\_RXNE**

Function Name      **\_\_STATIC\_INLINE void LL\_LPUART\_DisableIT\_RXNE  
(USART\_TypeDef \* LPUARTx)**

Function Description      Disable RX Not Empty Interrupt.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross
- CR1 RXNEIE LL\_LPUART\_DisableIT\_RXNE

reference:

### **LL\_LPUART\_DisableIT\_TC**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable Transmission Complete Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TCIE LL_LPUART_DisableIT_TC</li> </ul>

### **LL\_LPUART\_DisableIT\_TXE**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableIT_TXE (USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable TX Empty Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TXEIE LL_LPUART_DisableIT_TXE</li> </ul>

### **LL\_LPUART\_DisableIT\_PE**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable Parity Error Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PEIE LL_LPUART_DisableIT_PE</li> </ul>

### **LL\_LPUART\_DisableIT\_CM**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable Character Match Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CMIE LL_LPUART_DisableIT_CM</li> </ul>

**LL\_LPUART\_DisableIT\_ERROR**

Function Name	<code>__STATIC_INLINE void LL_LPUART_DisableIT_ERROR(     USART_TypeDef * LPUARTx)</code>
Function Description	Disable Error Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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 LPUARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 EIE LL_LPUART_DisableIT_ERROR</li> </ul>

**LL\_LPUART\_DisableIT\_CTS**

Function Name	<code>__STATIC_INLINE void LL_LPUART_DisableIT_CTS(     USART_TypeDef * LPUARTx)</code>
Function Description	Disable CTS Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSIE LL_LPUART_DisableIT_CTS</li> </ul>

**LL\_LPUART\_DisableIT\_WKUP**

Function Name	<code>__STATIC_INLINE void LL_LPUART_DisableIT_WKUP(     USART_TypeDef * LPUARTx)</code>
Function Description	Disable Wake Up from Stop Mode Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 WUFIE LL_LPUART_DisableIT_WKUP</li> </ul>

**LL\_LPUART\_IsEnabledIT\_IDLE**

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE(     USART_TypeDef * LPUARTx)</code>
Function Description	Check if the LPUART IDLE Interrupt source is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CR1 IDLEIE LL\_LPUART\_IsEnabledIT\_IDLE

### **LL\_LPUART\_IsEnabledIT\_RXNE**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART RX Not Empty Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 RXNEIE LL_LPUART_IsEnabledIT_RXNE</li> </ul>

### **LL\_LPUART\_IsEnabledIT\_TC**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Transmission Complete Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TCIE LL_LPUART_IsEnabledIT_TC</li> </ul>

### **LL\_LPUART\_IsEnabledIT\_TXE**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART TX Empty Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TXEIE LL_LPUART_IsEnabledIT_TXE</li> </ul>

### **LL\_LPUART\_IsEnabledIT\_PE**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE( USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if the LPUART Parity Error Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

- CR1 PEIE LL\_LPUART\_IsEnabledIT\_PE
- Reference Manual to  
LL API cross  
reference:

### **LL\_LPUART\_IsEnabledIT\_CM**

Function Name      **\_STATIC\_INLINE uint32\_t LL\_LPUART\_IsEnabledIT\_CM  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters      • **LPUARTx**: LPUART Instance

Return values      • **State**: of bit (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- CR1 CMIE LL\_LPUART\_IsEnabledIT\_CM

### **LL\_LPUART\_IsEnabledIT\_ERROR**

Function Name      **\_STATIC\_INLINE uint32\_t LL\_LPUART\_IsEnabledIT\_ERROR  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART Error Interrupt is enabled or disabled.

Parameters      • **LPUARTx**: LPUART Instance

Return values      • **State**: of bit (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- CR3 EIE LL\_LPUART\_IsEnabledIT\_ERROR

### **LL\_LPUART\_IsEnabledIT\_CTS**

Function Name      **\_STATIC\_INLINE uint32\_t LL\_LPUART\_IsEnabledIT\_CTS  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters      • **LPUARTx**: LPUART Instance

Return values      • **State**: of bit (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- CR3 CTSIE LL\_LPUART\_IsEnabledIT\_CTS

### **LL\_LPUART\_IsEnabledIT\_WKUP**

Function Name      **\_STATIC\_INLINE uint32\_t LL\_LPUART\_IsEnabledIT\_WKUP  
(USART\_TypeDef \* LPUARTx)**

Function Description      Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters      • **LPUARTx**: LPUART Instance

Return values      • **State**: of bit (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- CR3 WUFIE LL\_LPUART\_IsEnabledIT\_WKUP

### **LL\_LPUART\_EnableDMAReq\_RX**

Function Name      **`_STATIC_INLINE void LL_LPUART_EnableDMAReq_RX  
(USART_TypeDef * LPUARTx)`**

Function Description      Enable DMA Mode for reception.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR3 DMAR LL\_LPUART\_EnableDMAReq\_RX

### **LL\_LPUART\_DisableDMAReq\_RX**

Function Name      **`_STATIC_INLINE void LL_LPUART_DisableDMAReq_RX  
(USART_TypeDef * LPUARTx)`**

Function Description      Disable DMA Mode for reception.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR3 DMAR LL\_LPUART\_DisableDMAReq\_RX

### **LL\_LPUART\_IsEnabledDMAReq\_RX**

Function Name      **`_STATIC_INLINE uint32_t  
LL_LPUART_IsEnabledDMAReq_RX (USART_TypeDef *  
LPUARTx)`**

Function Description      Check if DMA Mode is enabled for reception.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **State:** of bit (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- CR3 DMAR LL\_LPUART\_IsEnabledDMAReq\_RX

### **LL\_LPUART\_EnableDMAReq\_TX**

Function Name      **`_STATIC_INLINE void LL_LPUART_EnableDMAReq_TX  
(USART_TypeDef * LPUARTx)`**

Function Description      Enable DMA Mode for transmission.

Parameters      • **LPUARTx:** LPUART Instance

Return values      • **None:**

- CR3 DMAT LL\_LPUART\_EnableDMAReq\_TX  
LL API cross reference:

### **LL\_LPUART\_DisableDMAReq\_TX**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable DMA Mode for transmission.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- CR3 DMAT LL\_LPUART\_DisableDMAReq\_TX  
LL API cross reference:

### **LL\_LPUART\_IsEnabledDMAReq\_TX**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (USART_TypeDef * LPUARTx)</code></b>
Function Description	Check if DMA Mode is enabled for transmission.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>

- CR3 DMAT LL\_LPUART\_IsEnabledDMAReq\_TX  
LL API cross reference:

### **LL\_LPUART\_EnableDMADeactOnRxErr**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)</code></b>
Function Description	Enable DMA Disabling on Reception Error.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- CR3 DDRE LL\_LPUART\_EnableDMADeactOnRxErr  
LL API cross reference:

### **LL\_LPUART\_DisableDMADeactOnRxErr**

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)</code></b>
Function Description	Disable DMA Disabling on Reception Error.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr</li> </ul>

### LL\_LPUART\_IsEnabledDMADeactOnRxErr

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (USART_TypeDef * LPUARTx)</code>
Function Description	Indicate if DMA Disabling on Reception Error is disabled.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr</li> </ul>

### LL\_LPUART\_DMA\_GetRegAddr

Function Name	<code>__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (USART_TypeDef * LPUARTx, uint32_t Direction)</code>
Function Description	Get the LPUART data register address used for DMA transfer.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> <li><b>Direction:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_LPUART_DMA_REG_DATA_TRANSMIT</li> <li>LL_LPUART_DMA_REG_DATA_RECEIVE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Address:</b> of data register</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>RDR RDR LL_LPUART_DMA_GetRegAddr</li> <li>TDR TDR LL_LPUART_DMA_GetRegAddr</li> </ul>

### LL\_LPUART\_ReceiveData8

Function Name	<code>__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (USART_TypeDef * LPUARTx)</code>
Function Description	Read Receiver Data register (Receive Data value, 8 bits)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Time:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>RDR RDR LL_LPUART_ReceiveData8</li> </ul>

### LL\_LPUART\_ReceiveData9

Function Name	<code>__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (USART_TypeDef * LPUARTx)</code>
---------------	--

Function Description	Read Receiver Data register (Receive Data value, 9 bits)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Time:</b> Value between Min_Data=0x00 and Max_Data=0x1FF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>RDR RDR LL_LPUART_ReceiveData9</li> </ul>

### LL\_LPUART\_TransmitData8

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_TransmitData8( USART_TypeDef * LPUARTx, uint8_t Value)</code></b>
Function Description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

Reference Manual to  
LL API cross  
reference:

### LL\_LPUART\_TransmitData9

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_TransmitData9( USART_TypeDef * LPUARTx, uint16_t Value)</code></b>
Function Description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0x1FF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

Reference Manual to  
LL API cross  
reference:

### LL\_LPUART\_RequestBreakSending

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_RequestBreakSending( USART_TypeDef * LPUARTx)</code></b>
Function Description	Request Break sending.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

Reference Manual to  
LL API cross  
reference:

### LL\_LPUART\_RequestEnterMuteMode

Function Name	<b><code>__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode( USART_TypeDef * LPUARTx)</code></b>
---------------	---

Function Description	Put LPUART in mute mode and set the RWU flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RQR MMRQ LL_LPUART_RequestEnterMuteMode</li> </ul>

### LL\_LPUART\_RequestRxDataFlush

Function Name	<b>STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)</b>
Function Description	Request a Receive Data flush.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RQR RXFRQ LL_LPUART_RequestRxDataFlush</li> </ul>

### LL\_LPUART\_DeInit

Function Name	<b>ErrorStatus LL_LPUART_DeInit (USART_TypeDef * LPUARTx)</b>
Function Description	De-initialize LPUART registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: LPUART registers are de-initialized</li> <li>– ERROR: not applicable</li> </ul> </li> </ul>

### LL\_LPUART\_Init

Function Name	<b>ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, LL_LPUART_InitTypeDef * LPUART_InitStruct)</b>
Function Description	Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTx:</b> LPUART Instance</li> <li>• <b>LPUART_InitStruct:</b> pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content</li> <li>– ERROR: Problem occurred during LPUART Registers initialization</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.</li> </ul>

- Baud rate value stored in LPUART\_InitStruct BaudRate field, should be valid (different from 0).

### **LL\_LPUART\_StructInit**

Function Name	<b>void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)</b>
Function Description	Set each LL_LPUART_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUART_InitStruct:</b> pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## **67.3 LPUART Firmware driver defines**

### **67.3.1 LPUART**

#### ***Address Length Detection***

**LL\_LPUART\_ADDRESS\_DETECT\_4B** 4-bit address detection method selected

**LL\_LPUART\_ADDRESS\_DETECT\_7B** 7-bit address detection (in 8-bit data mode) method selected

#### ***Binary Data Inversion***

**LL\_LPUART\_BINARY\_LOGIC\_POSITIVE** Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

**LL\_LPUART\_BINARY\_LOGIC\_NEGATIVE** Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

#### ***Bit Order***

**LL\_LPUART\_BITORDER\_LSBFIRST** data is transmitted/received with data bit 0 first, following the start bit

**LL\_LPUART\_BITORDER\_MSBFIRST** data is transmitted/received with the MSB first, following the start bit

#### ***Clear Flags Defines***

**LL\_LPUART\_ICR\_PECF** Parity error flag

**LL\_LPUART\_ICR\_FECF** Framing error flag

**LL\_LPUART\_ICR\_NCF** Noise detected flag

**LL\_LPUART\_ICR\_ORECF** Overrun error flag

**LL\_LPUART\_ICR\_IDLECF** Idle line detected flag

**LL\_LPUART\_ICR\_TCCF** Transmission complete flag

**LL\_LPUART\_ICR\_CTSCF** CTS flag

**LL\_LPUART\_ICR\_CMCF** Character match flag

**LL\_LPUART\_ICR\_WUCF** Wakeup from Stop mode flag

#### ***Datawidth***

`LL_LPUART_DATAWIDTH_7B` 7 bits word length : Start bit, 7 data bits, n stop bits

`LL_LPUART_DATAWIDTH_8B` 8 bits word length : Start bit, 8 data bits, n stop bits

`LL_LPUART_DATAWIDTH_9B` 9 bits word length : Start bit, 9 data bits, n stop bits

#### ***Driver Enable Polarity***

`LL_LPUART_DE_POLARITY_HIGH` DE signal is active high

`LL_LPUART_DE_POLARITY_LOW` DE signal is active low

#### ***Direction***

`LL_LPUART_DIRECTION_NONE` Transmitter and Receiver are disabled

`LL_LPUART_DIRECTION_RX` Transmitter is disabled and Receiver is enabled

`LL_LPUART_DIRECTION_TX` Transmitter is enabled and Receiver is disabled

`LL_LPUART_DIRECTION_TX_RX` Transmitter and Receiver are enabled

#### ***DMA Register Data***

`LL_LPUART_DMA_REG_DATA_TRANSMIT` Get address of data register used for transmission

`LL_LPUART_DMA_REG_DATA_RECEIVE` Get address of data register used for reception

#### ***Get Flags Defines***

`LL_LPUART_ISR_PE` Parity error flag

`LL_LPUART_ISR_FE` Framing error flag

`LL_LPUART_ISR_NE` Noise detected flag

`LL_LPUART_ISR_ORE` Overrun error flag

`LL_LPUART_ISR_IDLE` Idle line detected flag

`LL_LPUART_ISR_RXNE` Read data register not empty flag

`LL_LPUART_ISR_TC` Transmission complete flag

`LL_LPUART_ISR_TXE` Transmit data register empty flag

`LL_LPUART_ISR_CTSIF` CTS interrupt flag

`LL_LPUART_ISR_CTS` CTS flag

`LL_LPUART_ISR_BUSY` Busy flag

`LL_LPUART_ISR_CMF` Character match flag

`LL_LPUART_ISR_SBKF` Send break flag

`LL_LPUART_ISR_RWU` Receiver wakeup from Mute mode flag

`LL_LPUART_ISR_WUF` Wakeup from Stop mode flag

`LL_LPUART_ISR_TEACK` Transmit enable acknowledge flag

`LL_LPUART_ISR_REACK` Receive enable acknowledge flag

#### ***Hardware Control***

`LL_LPUART_HWCONTROL_NONE` CTS and RTS hardware flow control disabled

`LL_LPUART_HWCONTROL_RTS` RTS output enabled, data is only requested

LL_LPUART_HWCONTROL_CTS	when there is space in the receive buffer CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)
LL_LPUART_HWCONTROL_RTS_CTS	CTS and RTS hardware flow control enabled

**IT Defines**

LL_LPUART_CR1_IDLEIE	IDLE interrupt enable
LL_LPUART_CR1_RXNEIE	Read data register not empty interrupt enable
LL_LPUART_CR1_TCIE	Transmission complete interrupt enable
LL_LPUART_CR1_TXEIE	Transmit data register empty interrupt enable
LL_LPUART_CR1_PEIE	Parity error
LL_LPUART_CR1_CMIE	Character match interrupt enable
LL_LPUART_CR3_EIE	Error interrupt enable
LL_LPUART_CR3_CTSIE	CTS interrupt enable
LL_LPUART_CR3_WUFIE	Wakeup from Stop mode interrupt enable

**Parity Control**

LL_LPUART_PARITY_NONE	Parity control disabled
LL_LPUART_PARITY EVEN	Parity control enabled and Even Parity is selected
LL_LPUART_PARITY ODD	Parity control enabled and Odd Parity is selected

**RX Pin Active Level Inversion**

LL_LPUART_RXPIN_LEVEL_STANDARD	RX pin signal works using the standard logic levels
LL_LPUART_RXPIN_LEVEL_INVERTED	RX pin signal values are inverted.

**Stop Bits**

LL_LPUART_STOPBITS_1	1 stop bit
LL_LPUART_STOPBITS_2	2 stop bits

**TX Pin Active Level Inversion**

LL_LPUART_TXPIN_LEVEL_STANDARD	TX pin signal works using the standard logic levels
LL_LPUART_TXPIN_LEVEL_INVERTED	TX pin signal values are inverted.

**TX RX Pins Swap**

LL_LPUART_TXRX_STANDARD	TX/RX pins are used as defined in standard pinout
LL_LPUART_TXRX_SWAPPED	TX and RX pins functions are swapped.

**Wakeup**

LL_LPUART_WAKEUP_IDLELINE	LPUART wake up from Mute mode on Idle Line
LL_LPUART_WAKEUP_ADDRESSMARK	LPUART wake up from Mute mode on Address Mark

**Wakeup Activation**

---

<code>LL_LPUART_WAKEUP_ON_ADDRESS</code>	Wake up active on address match
<code>LL_LPUART_WAKEUP_ON_STARTBIT</code>	Wake up active on Start bit detection
<code>LL_LPUART_WAKEUP_ON_RXNE</code>	Wake up active on RXNE

**Helper Macros**

<code>_LL_LPUART_DIV</code>	<b>Description:</b>
-----------------------------	---------------------

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

**Parameters:**

- `_PERIPHCLK_`: Peripheral Clock frequency used for LPUART Instance
- `_BAUDRATE_`: Baud Rate value to achieve

**Return value:**

- LPUARTDIV: value to be used for BRR register filling

**Common Write and read registers Macros**

<code>LL_LPUART_WriteReg</code>	<b>Description:</b>
---------------------------------	---------------------

- Write a value in LPUART register.

**Parameters:**

- `_INSTANCE_`: LPUART Instance
- `_REG_`: Register to be written
- `_VALUE_`: Value to be written in the register

**Return value:**

- None

<code>LL_LPUART_ReadReg</code>	<b>Description:</b>
--------------------------------	---------------------

- Read a value in LPUART register.

**Parameters:**

- `_INSTANCE_`: LPUART Instance
- `_REG_`: Register to be read

**Return value:**

- Register: value

## 68 LL PWR Generic Driver

### 68.1 PWR Firmware driver API description

#### 68.1.1 Detailed description of functions

##### **LL\_PWR\_EnableLowPowerRunMode**

Function Name	<b><code>__STATIC_INLINE void LL_PWR_EnableLowPowerRunMode(void)</code></b>
Function Description	Switch the regulator from main mode to low-power mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Remind to set the regulator to low power before enabling LowPower run mode (bit <code>LL_PWR_REGU_LPMODES_LOW_POWER</code>).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPRUN <code>LL_PWR_EnableLowPowerRunMode</code></li> </ul>

##### **LL\_PWR\_DisableLowPowerRunMode**

Function Name	<b><code>__STATIC_INLINE void LL_PWR_DisableLowPowerRunMode(void)</code></b>
Function Description	Switch the regulator from low-power mode to main mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPRUN <code>LL_PWR_DisableLowPowerRunMode</code></li> </ul>

##### **LL\_PWR\_IsEnabledLowPowerRunMode**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledLowPowerRunMode(void)</code></b>
Function Description	Check if the regulator is in low-power mode.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPRUN <code>LL_PWR_IsEnabledLowPowerRunMode</code></li> </ul>

##### **LL\_PWR\_EnterLowPowerRunMode**

Function Name	<b><code>__STATIC_INLINE void LL_PWR_EnterLowPowerRunMode(void)</code></b>
Description	Set voltage regulator to low-power and switch from run main mode to run low-power mode.

tion	
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This "high level" function is introduced to provide functional compatibility with other families. Notice that the two registers have to be written sequentially, so this function is not atomic. To assure atomicity you can call separately the following functions: LL_PWR_SetRegulModeLP(LL_PWR_REGU_LPMODES_LOW_POWER); LL_PWR_EnableLowPowerRunMode();</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPSDSR LL_PWR_EnterLowPowerRunMode</li> <li>• CR LPRUN LL_PWR_EnterLowPowerRunMode</li> </ul>

### LL\_PWR\_ExitLowPowerRunMode

Function Name	<b><code>_STATIC_INLINE void LL_PWR_ExitLowPowerRunMode (void )</code></b>
Function Description	Set voltage regulator to main and switch from run main mode to low-power mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This "high level" function is introduced to provide functional compatibility with other families. Notice that the two registers have to be written sequentially, so this function is not atomic. To assure atomicity you can call separately the following functions: LL_PWR_DisableLowPowerRunMode();LL_PWR_SetRegulModeLP(LL_PWR_REGU_LPMODES_MAIN);</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPSDSR LL_PWR_ExitLowPowerRunMode</li> <li>• CR LPRUN LL_PWR_ExitLowPowerRunMode</li> </ul>

### LL\_PWR\_SetRegulVoltageScaling

Function Name	<b><code>_STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)</code></b>
Function Description	Set the main internal regulator output voltage.
Parameters	<ul style="list-style-type: none"> <li>• <b>VoltageScaling:</b> This parameter can be one of the following values:</li> </ul>

	<ul style="list-style-type: none"> <li>- LL_PWR_REGU_VOLTAGE_SCALE1</li> <li>- LL_PWR_REGU_VOLTAGE_SCALE2</li> <li>- LL_PWR_REGU_VOLTAGE_SCALE3</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR VOS LL_PWR_SetRegulVoltageScaling</li> </ul>

### LL\_PWR\_GetRegulVoltageScaling

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling(void)</code>
Function Description	Get the main internal regulator output voltage.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_PWR_REGU_VOLTAGE_SCALE1</li> <li>- LL_PWR_REGU_VOLTAGE_SCALE2</li> <li>- LL_PWR_REGU_VOLTAGE_SCALE3</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR VOS LL_PWR_GetRegulVoltageScaling</li> </ul>

### LL\_PWR\_EnableBkUpAccess

Function Name	<code>__STATIC_INLINE void LL_PWR_EnableBkUpAccess(void)</code>
Function Description	Enable access to the backup domain.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR DBP LL_PWR_EnableBkUpAccess</li> </ul>

### LL\_PWR\_DisableBkUpAccess

Function Name	<code>__STATIC_INLINE void LL_PWR_DisableBkUpAccess(void)</code>
Function Description	Disable access to the backup domain.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR DBP LL_PWR_DisableBkUpAccess</li> </ul>

### LL\_PWR\_IsEnabledBkUpAccess

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess(void)</code>
Function Description	Check if the backup domain is enabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR DBP LL_PWR_IsEnabledBkUpAccess</li> </ul>

reference:

### **LL\_PWR\_SetRegulModeLP**

Function Name	<b><code>_STATIC_INLINE void LL_PWR_SetRegulModeLP (uint32_t RegulMode)</code></b>
Function Description	Set voltage regulator mode during low power modes.
Parameters	<ul style="list-style-type: none"> <li>• <b>RegulMode:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_PWR_REGU_LPMODES_MAIN</code></li> <li>- <code>LL_PWR_REGU_LPMODES_LOW_POWER</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPSDSR LL_PWR_SetRegulModeLP</li> </ul>

### **LL\_PWR\_GetRegulModeLP**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_PWR_GetRegulModeLP (void )</code></b>
Function Description	Get voltage regulator mode during low power modes.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_PWR_REGU_LPMODES_MAIN</code></li> <li>- <code>LL_PWR_REGU_LPMODES_LOW_POWER</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR LPSDSR LL_PWR_GetRegulModeLP</li> </ul>

### **LL\_PWR\_SetPowerMode**

Function Name	<b><code>_STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t PDMode)</code></b>
Function Description	Set power down mode when CPU enters deepsleep.
Parameters	<ul style="list-style-type: none"> <li>• <b>PDMode:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_PWR_MODE_STOP</code></li> <li>- <code>LL_PWR_MODE_STANDBY</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Set the regulator to low power (bit <code>LL_PWR_REGU_LPMODES_LOW_POWER</code>) before setting <code>MODE_STOP</code>. If the regulator remains in "main mode", it consumes more power without providing any additional feature. In <code>MODE_STANDBY</code> the regulator is automatically off.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PDSS LL_PWR_SetPowerMode</li> </ul>

**LL\_PWR\_GetPowerMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void )</code>
Function Description	Get power down mode when CPU enters deepsleep.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_PWR_MODE_STOP</code></li> <li>- <code>LL_PWR_MODE_STANDBY</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PDDS LL_PWR_GetPowerMode</li> </ul>

**LL\_PWR\_SetPVDLevel**

Function Name	<code>__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)</code>
Function Description	Configure the voltage threshold detected by the Power Voltage Detector.
Parameters	<ul style="list-style-type: none"> <li>• <b>PVDLevel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_PWR_PVDLEVEL_0</code></li> <li>- <code>LL_PWR_PVDLEVEL_1</code></li> <li>- <code>LL_PWR_PVDLEVEL_2</code></li> <li>- <code>LL_PWR_PVDLEVEL_3</code></li> <li>- <code>LL_PWR_PVDLEVEL_4</code></li> <li>- <code>LL_PWR_PVDLEVEL_5</code></li> <li>- <code>LL_PWR_PVDLEVEL_6</code></li> <li>- <code>LL_PWR_PVDLEVEL_7</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PLS LL_PWR_SetPVDLevel</li> </ul>

**LL\_PWR\_GetPVDLevel**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void )</code>
Function Description	Get the voltage threshold detection.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_PWR_PVDLEVEL_0</code></li> <li>- <code>LL_PWR_PVDLEVEL_1</code></li> <li>- <code>LL_PWR_PVDLEVEL_2</code></li> <li>- <code>LL_PWR_PVDLEVEL_3</code></li> <li>- <code>LL_PWR_PVDLEVEL_4</code></li> <li>- <code>LL_PWR_PVDLEVEL_5</code></li> <li>- <code>LL_PWR_PVDLEVEL_6</code></li> <li>- <code>LL_PWR_PVDLEVEL_7</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PLS LL_PWR_GetPVDLevel</li> </ul>

**LL\_PWR\_EnablePVD**

Function Name	<code>__STATIC_INLINE void LL_PWR_EnablePVD (void )</code>
Function Description	Enable Power Voltage Detector.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PVDE LL_PWR_EnablePVD</li> </ul>

**LL\_PWR\_DisablePVD**

Function Name	<code>__STATIC_INLINE void LL_PWR_DisablePVD (void )</code>
Function Description	Disable Power Voltage Detector.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PVDE LL_PWR_DisablePVD</li> </ul>

**LL\_PWR\_IsEnabledPVD**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void )</code>
Function Description	Check if Power Voltage Detector is enabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR PVDE LL_PWR_IsEnabledPVD</li> </ul>

**LL\_PWR\_EnableWakeUpPin**

Function Name	<code>__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)</code>
Function Description	Enable the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPin:</b> This parameter can be one of the following values: (*) not available on all devices <ul style="list-style-type: none"> <li>- LL_PWR_WAKEUP_PIN1</li> <li>- LL_PWR_WAKEUP_PIN2</li> <li>- LL_PWR_WAKEUP_PIN3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR EWUP1 LL_PWR_EnableWakeUpPin</li> <li>• CSR EWUP2 LL_PWR_EnableWakeUpPin</li> <li>• CSR EWUP3 LL_PWR_EnableWakeUpPin</li> </ul>

**LL\_PWR\_DisableWakeUpPin**

Function Name	<code>__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)</code>
Function Description	Disable the WakeUp PINx functionality.

Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPin:</b> This parameter can be one of the following values: (*) not available on all devices           <ul style="list-style-type: none"> <li>- LL_PWR_WAKEUP_PIN1</li> <li>- LL_PWR_WAKEUP_PIN2</li> <li>- LL_PWR_WAKEUP_PIN3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR EWUP1 LL_PWR_DisableWakeUpPin</li> <li>• CSR EWUP2 LL_PWR_DisableWakeUpPin</li> <li>• CSR EWUP3 LL_PWR_DisableWakeUpPin</li> </ul>

### LL\_PWR\_IsEnabledWakeUpPin

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin(           uint32_t WakeUpPin)</code>
Function Description	Check if the WakeUp PINx functionality is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPin:</b> This parameter can be one of the following values: (*) not available on all devices           <ul style="list-style-type: none"> <li>- LL_PWR_WAKEUP_PIN1</li> <li>- LL_PWR_WAKEUP_PIN2</li> <li>- LL_PWR_WAKEUP_PIN3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR EWUP1 LL_PWR_IsEnabledWakeUpPin</li> <li>• CSR EWUP2 LL_PWR_IsEnabledWakeUpPin</li> <li>• CSR EWUP3 LL_PWR_IsEnabledWakeUpPin</li> </ul>

### LL\_PWR\_EnableUltraLowPower

Function Name	<code>__STATIC_INLINE void LL_PWR_EnableUltraLowPower(void )</code>
Function Description	Enable ultra low-power mode by enabling VREFINT switch off in low-power modes.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ULP LL_PWR_EnableUltraLowPower</li> </ul>

### LL\_PWR\_DisableUltraLowPower

Function Name	<code>__STATIC_INLINE void LL_PWR_DisableUltraLowPower(void )</code>
Function Description	Disable ultra low-power mode by disabling VREFINT switch off in low-power modes.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ULP LL_PWR_DisableUltraLowPower</li> </ul>

**LL\_PWR\_IsEnabledUltraLowPower**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledUltraLowPower(void)</code>
Function Description	Check if ultra low-power mode is enabled by checking if VREFINT switch off in low-power modes is enabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ULP LL_PWR_IsEnabledUltraLowPower</li> </ul>

**LL\_PWR\_EnableFastWakeUp**

Function Name	<code>__STATIC_INLINE void LL_PWR_EnableFastWakeUp(void)</code>
Function Description	Enable fast wakeup by ignoring VREFINT startup time when exiting from low-power mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Works in conjunction with ultra low power mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR FWU LL_PWR_EnableFastWakeUp</li> </ul>

**LL\_PWR\_DisableFastWakeUp**

Function Name	<code>__STATIC_INLINE void LL_PWR_DisableFastWakeUp(void)</code>
Function Description	Disable fast wakeup by waiting VREFINT startup time when exiting from low-power mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Works in conjunction with ultra low power mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR FWU LL_PWR_DisableFastWakeUp</li> </ul>

**LL\_PWR\_IsEnabledFastWakeUp**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledFastWakeUp(void)</code>
Function Description	Check if fast wakeup is enabled by checking if VREFINT startup time when exiting from low-power mode is ignored.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR FWU LL_PWR_IsEnabledFastWakeUp</li> </ul>

**LL\_PWR\_EnableNVMKeptOff**

Function Name	<code>__STATIC_INLINE void LL_PWR_EnableNVMKeptOff(void)</code>
---------------	---

Function Description	Enable non-volatile memory (Flash and EEPROM) keeping off feature when exiting from low-power mode.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When enabled, after entering low-power mode (Stop or Standby only), if RUN_PD of FLASH_ACR register is also set, the Flash memory will not be woken up when exiting from deepsleep mode. When enabled, the EEPROM will not be woken up when exiting from low-power mode (if the bit RUN_PD is set)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR DS_EE_KOFF LL_PWR_EnableNVMKeptOff</li> </ul>

### LL\_PWR\_DisableNVMKeptOff

Function Name	<code>__STATIC_INLINE void LL_PWR_DisableNVMKeptOff (void )</code>
Function Description	Disable non-volatile memory (Flash and EEPROM) keeping off feature when exiting from low-power mode.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When disabled, Flash memory is woken up when exiting from deepsleep mode even if the bit RUN_PD is set</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR DS_EE_KOFF LL_PWR_DisableNVMKeptOff</li> </ul>

### LL\_PWR\_IsEnabledNVMKeptOff

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsEnabledNVMKeptOff (void )</code>
Function Description	Check if non-volatile memory (Flash and EEPROM) keeping off feature when exiting from low-power mode is enabled.
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR DS_EE_KOFF LL_PWR_IsEnabledNVMKeptOff</li> </ul>

### LL\_PWR\_IsActiveFlag\_WU

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU (void )</code>
Function Description	Get Wake-up Flag.
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CSR WUF LL_PWR_IsActiveFlag_WU</li> </ul>

**LL\_PWR\_IsActiveFlag\_SB**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void )</code>
Function Description	Get Standby Flag.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR SBF LL_PWR_IsActiveFlag_SB</li> </ul>

**LL\_PWR\_IsActiveFlag\_PVDO**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void )</code>
Function Description	Indicate whether VDD voltage is below the selected PVD threshold.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR PVDO LL_PWR_IsActiveFlag_PVDO</li> </ul>

**LL\_PWR\_IsActiveFlag\_VREFINTRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VREFINTRDY (void )</code>
Function Description	Get Internal Reference VrefInt Flag.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR VREFINTRDYF LL_PWR_IsActiveFlag_VREFINTRDY</li> </ul>

**LL\_PWR\_IsActiveFlag\_VOSF**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOSF (void )</code>
Function Description	Indicate whether the regulator is ready in the selected voltage range or if its output voltage is still changing to the required voltage level.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR VOSF LL_PWR_IsActiveFlag_VOSF</li> </ul>

**LL\_PWR\_IsActiveFlag\_REGLPF**

Function Name	<code>__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPF (void )</code>
Function Description	Indicate whether the regulator is ready in main mode or is in low-power mode.

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Take care, return value "0" means the regulator is ready. Return value "1" means the output voltage range is still changing.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CSR REGLPF LL_PWR_IsActiveFlag_REGLPF</li> </ul>

### LL\_PWR\_ClearFlag\_SB

Function Name	<code>__STATIC_INLINE void LL_PWR_ClearFlag_SB (void )</code>
Function Description	Clear Standby Flag.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR CSBF LL_PWR_ClearFlag_SB</li> </ul>

### LL\_PWR\_ClearFlag\_WU

Function Name	<code>__STATIC_INLINE void LL_PWR_ClearFlag_WU (void )</code>
Function Description	Clear Wake-up Flags.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR CWUF LL_PWR_ClearFlag_WU</li> </ul>

### LL\_PWR\_DelInit

Function Name	<code>ErrorStatus LL_PWR_DelInit (void )</code>
Function Description	De-initialize the PWR registers to their default reset values.
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: PWR registers are de-initialized</li> <li>ERROR: not applicable</li> </ul> </li> </ul>

## 68.2 PWR Firmware driver defines

### 68.2.1 PWR

#### *Clear Flags Defines*

<code>LL_PWR_CR_CSBF</code>	Clear standby flag
<code>LL_PWR_CR_CWUF</code>	Clear wakeup flag

#### *Get Flags Defines*

<code>LL_PWR_CSR_WUF</code>	Wakeup flag
<code>LL_PWR_CSR_SBF</code>	Standby flag
<code>LL_PWR_CSR_PVDO</code>	Power voltage detector output flag

---

LL_PWR_CSR_VREFINTRDYF	VREFINT ready flag
LL_PWR_CSR_VOSF	Voltage scaling select flag
LL_PWR_CSR_REGLPF	Regulator low power flag
LL_PWR_CSR_EWUP1	Enable WKUP pin 1
LL_PWR_CSR_EWUP2	Enable WKUP pin 2
LL_PWR_CSR_EWUP3	Enable WKUP pin 3

**Mode Power**

LL_PWR_MODE_STOP	Enter Stop mode 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 1.9 V
LL_PWR_PVDLEVEL_1	Voltage threshold detected by PVD 2.1 V
LL_PWR_PVDLEVEL_2	Voltage threshold detected by PVD 2.3 V
LL_PWR_PVDLEVEL_3	Voltage threshold detected by PVD 2.5 V
LL_PWR_PVDLEVEL_4	Voltage threshold detected by PVD 2.7 V
LL_PWR_PVDLEVEL_5	Voltage threshold detected by PVD 2.9 V
LL_PWR_PVDLEVEL_6	Voltage threshold detected by PVD 3.1 V
LL_PWR_PVDLEVEL_7	External input analog voltage (Compare internally to VREFINT)

**Regulator Mode In Low Power Modes**

LL_PWR_REGU_LPMODES_MAIN	Voltage regulator in main mode during deepsleep/sleep/low-power run mode
LL_PWR_REGU_LPMODES_LOW_POWER	Voltage regulator in low-power mode during deepsleep/sleep/low-power run mode

**Regulator Voltage**

LL_PWR_REGU_VOLTAGE_SCALE1	1.8V (range 1)
LL_PWR_REGU_VOLTAGE_SCALE2	1.5V (range 2)
LL_PWR_REGU_VOLTAGE_SCALE3	1.2V (range 3)

**Wakeup Pins**

LL_PWR_WAKEUP_PIN1	WKUP pin 1 : PA0
LL_PWR_WAKEUP_PIN2	WKUP pin 2 : PC13
LL_PWR_WAKEUP_PIN3	WKUP pin 3 : PE6 or PA2 according to device

**Common write and read registers Macros**

LL_PWR_WriteReg	Description:
	<ul style="list-style-type: none"> <li>Write a value in PWR register.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li><u>__REG__</u>: Register to be written</li> <li><u>__VALUE__</u>: Value to be written in the register</li> </ul>

**Return value:**

- None

`LL_PWR_ReadReg`

**Description:**

- Read a value in PWR register.

**Parameters:**

- `__REG__`: Register to be read

**Return value:**

- Register: value

## 69 LL RCC Generic Driver

### 69.1 RCC Firmware driver registers structures

#### 69.1.1 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

### 69.2 RCC Firmware driver API description

#### 69.2.1 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:

##### 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:

**LL\_RCC\_HSE\_DisableBypass**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void )</code>
Function Description	Disable HSE external oscillator (HSE Bypass)
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR HSEBYP LL_RCC_HSE_DisableBypass</li> </ul>

**LL\_RCC\_HSE\_Enable**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSE_Enable (void )</code>
Function Description	Enable HSE crystal oscillator (HSE ON)
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR HSEON LL_RCC_HSE_Enable</li> </ul>

**LL\_RCC\_HSE\_Disable**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSE_Disable (void )</code>
Function Description	Disable HSE crystal oscillator (HSE ON)
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR HSEON LL_RCC_HSE_Disable</li> </ul>

**LL\_RCC\_HSE\_IsReady**

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void )</code>
Function Description	Check if HSE oscillator Ready.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR HSERDY LL_RCC_HSE_IsReady</li> </ul>

**LL\_RCC\_SetRTC\_HSEPrescaler**

Function Name	<code>__STATIC_INLINE void LL_RCC_SetRTC_HSEPrescaler (uint32_t Div)</code>
Function Description	Configure the RTC prescaler (divider)
Parameters	<ul style="list-style-type: none"> <li>• <b>Div:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_RCC_RTC_HSE_DIV_2</code></li> <li>- <code>LL_RCC_RTC_HSE_DIV_4</code></li> <li>- <code>LL_RCC_RTC_HSE_DIV_8</code></li> <li>- <code>LL_RCC_RTC_HSE_DIV_16</code></li> </ul> </li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR RTCPRE LL_RCC_SetRTC_HSEPrescaler</li> </ul>

### LL\_RCC\_GetRTC\_HSEPrescaler

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_GetRTC_HSEPrescaler(void)</code>
Function Description	Get the RTC divider (prescaler)
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_RTC_HSE_DIV_2</li> <li>- LL_RCC_RTC_HSE_DIV_4</li> <li>- LL_RCC_RTC_HSE_DIV_8</li> <li>- LL_RCC_RTC_HSE_DIV_16</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR RTCPRE LL_RCC_GetRTC_HSEPrescaler</li> </ul>

### LL\_RCC\_HSI\_Enable

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_Enable(void)</code>
Function Description	Enable HSI oscillator.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR HSION LL_RCC_HSI_Enable</li> </ul>

### LL\_RCC\_HSI\_Disable

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_Disable(void)</code>
Function Description	Disable HSI oscillator.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR HSION LL_RCC_HSI_Disable</li> </ul>

### LL\_RCC\_HSI\_IsReady

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady(void)</code>
Function Description	Check if HSI clock is ready.
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR HSIRDY LL_RCC_HSI_IsReady</li> </ul>

**LL\_RCC\_HSI\_EnableInStopMode**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void )</code>
Function Description	Enable HSI even in stop mode.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• HSI oscillator is forced ON even in Stop mode</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR HSIKERON LL_RCC_HSI_EnableInStopMode</li></ul>

**LL\_RCC\_HSI\_DisableInStopMode**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void )</code>
Function Description	Disable HSI in stop mode.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR HSIKERON LL_RCC_HSI_DisableInStopMode</li></ul>

**LL\_RCC\_HSI\_EnableDivider**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_EnableDivider (void )</code>
Function Description	Enable HSI Divider (it divides by 4)
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR HSIDIVEN LL_RCC_HSI_EnableDivider</li></ul>

**LL\_RCC\_HSI\_DisableDivider**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_DisableDivider (void )</code>
Function Description	Disable HSI Divider (it divides by 4)
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR HSIDIVEN LL_RCC_HSI_DisableDivider</li></ul>

**LL\_RCC\_HSI\_EnableOutput**

Function Name	<code>__STATIC_INLINE void LL_RCC_HSI_EnableOutput (void )</code>
Function Description	Enable HSI Output.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"><li>• CR HSIOUTEN LL_RCC_HSI_EnableOutput</li></ul>

reference:

### **LL\_RCC\_HSI\_DisableOutput**

Function Name      **`_STATIC_INLINE void LL_RCC_HSI_DisableOutput (void )`**

Function Description      Disable HSI Output.

Return values      •    **None:**

Reference Manual to  
LL API cross  
reference:

- CR HSIOUTEN LL\_RCC\_HSI\_DisableOutput

### **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:

- ICSCR 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:

- ICSCR 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:

### LL\_RCC\_HSI48\_Enable

Function Name **`__STATIC_INLINE void LL_RCC_HSI48_Enable (void )`**

Function Description Enable HSI48.

Return values • **None:**

Reference Manual to CRRCR HSI48ON LL\_RCC\_HSI48\_Enable  
LL API cross reference:

### LL\_RCC\_HSI48\_Disable

Function Name **`__STATIC_INLINE void LL_RCC_HSI48_Disable (void )`**

Function Description Disable HSI48.

Return values • **None:**

Reference Manual to CRRCR HSI48ON LL\_RCC\_HSI48\_Disable  
LL API cross reference:

### LL\_RCC\_HSI48\_IsReady

Function Name **`__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void )`**

Function Description Check if HSI48 oscillator Ready.

Return values • **State:** of bit (1 or 0).

Reference Manual to CRRCR HSI48RDY LL\_RCC\_HSI48\_IsReady  
LL API cross reference:

### LL\_RCC\_HSI48\_GetCalibration

Function Name **`__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void )`**

Function Description Get HSI48 Calibration value.

Return values • **Between:** Min\_Data = 0x00 and Max\_Data = 0xFF

Reference Manual to CRRCR HSI48CAL LL\_RCC\_HSI48\_GetCalibration  
LL API cross reference:

### LL\_RCC\_HSI48\_EnableDivider

Function Name **`__STATIC_INLINE void LL_RCC_HSI48_EnableDivider (void )`**

Function Description Enable HSI48 Divider (it divides by 6)

Return values • **None:**

Reference Manual to CRRCR HSI48DIV6OUTEN LL\_RCC\_HSI48\_EnableDivider  
LL API cross

reference:

### **LL\_RCC\_HSI48\_DisableDivider**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_HSI48_DisableDivider (void )</code></b>
Function Description	Disable HSI48 Divider (it divides by 6)
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CRRCR HSI48DIV6OUTEN LL_RCC_HSI48_DisableDivider</li> </ul>

### **LL\_RCC\_HSI48\_IsDivided**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_HSI48_IsDivided (void )</code></b>
Function Description	Check if HSI48 Divider is enabled (it divides by 6)
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CRRCR HSI48DIV6OUTEN LL_RCC_HSI48_IsDivided</li> </ul>

### **LL\_RCC\_LSE\_Enable**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_Enable (void )</code></b>
Function Description	Enable Low Speed External (LSE) crystal.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSEON LL_RCC_LSE_Enable</li> </ul>

### **LL\_RCC\_LSE\_Disable**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_Disable (void )</code></b>
Function Description	Disable Low Speed External (LSE) crystal.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSEON LL_RCC_LSE_Disable</li> </ul>

### **LL\_RCC\_LSE\_EnableBypass**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void )</code></b>
Function Description	Enable external clock source (LSE bypass).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSEBYP LL_RCC_LSE_EnableBypass</li> </ul>

**LL\_RCC\_LSE\_DisableBypass**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void )</code></b>
Function Description	Disable external clock source (LSE bypass).
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSEBYP LL_RCC_LSE_DisableBypass</li> </ul>

**LL\_RCC\_LSE\_SetDriveCapability**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)</code></b>
Function Description	Set LSE oscillator drive capability.
Parameters	<ul style="list-style-type: none"> <li>• <b>LSEDrive:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RCC_LSEDRIVE_LOW</li> <li>– LL_RCC_LSEDRIVE_MEDIUMLOW</li> <li>– LL_RCC_LSEDRIVE_MEDIUMHIGH</li> <li>– LL_RCC_LSEDRIVE_HIGH</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The oscillator is in Xtal mode when it is not in bypass mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSEDRV LL_RCC_LSE_SetDriveCapability</li> </ul>

**LL\_RCC\_LSE\_GetDriveCapability**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void )</code></b>
Function Description	Get LSE oscillator drive capability.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RCC_LSEDRIVE_LOW</li> <li>– LL_RCC_LSEDRIVE_MEDIUMLOW</li> <li>– LL_RCC_LSEDRIVE_MEDIUMHIGH</li> <li>– LL_RCC_LSEDRIVE_HIGH</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSEDRV LL_RCC_LSE_GetDriveCapability</li> </ul>

**LL\_RCC\_LSE\_EnableCSS**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void )</code></b>
Function Description	Enable Clock security system on LSE.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CSR LSECSSON LL_RCC_LSE_EnableCSS</li> </ul>

reference:

### **LL\_RCC\_LSE\_DisableCSS**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void )</code></b>
Function Description	Disable Clock security system on LSE.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Clock security system can be disabled only after a LSE failure detection. In that case it MUST be disabled by software.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSECSSON LL_RCC_LSE_DisableCSS</li> </ul>

### **LL\_RCC\_LSE\_IsReady**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void )</code></b>
Function Description	Check if LSE oscillator Ready.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSERDY LL_RCC_LSE_IsReady</li> </ul>

### **LL\_RCC\_LSE\_IsCSSDetected**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_LSE_IsCSSDetected (void )</code></b>
Function Description	Check if CSS on LSE failure Detection.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSECSSD LL_RCC_LSE_IsCSSDetected</li> </ul>

### **LL\_RCC\_LSI\_Enable**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSI_Enable (void )</code></b>
Function Description	Enable LSI Oscillator.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR LSION LL_RCC_LSI_Enable</li> </ul>

### **LL\_RCC\_LSI\_Disable**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_LSI_Disable (void )</code></b>
Function Description	Disable LSI Oscillator.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to • CSR LSION LL\_RCC\_LSI\_Disable  
LL API cross reference:

### LL\_RCC\_LSI\_IsReady

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)</code>
Function Description	Check if LSI is Ready.
Return values	• <b>State:</b> of bit (1 or 0).
Reference Manual to LL API cross reference:	• CSR LSIRDY LL_RCC_LSI_IsReady

### LL\_RCC\_MSI\_Enable

Function Name	<code>__STATIC_INLINE void LL_RCC_MSI_Enable (void)</code>
Function Description	Enable MSI oscillator.
Return values	• <b>None:</b>
Reference Manual to LL API cross reference:	• CR MSION LL_RCC_MSI_Enable

### LL\_RCC\_MSI\_Disable

Function Name	<code>__STATIC_INLINE void LL_RCC_MSI_Disable (void)</code>
Function Description	Disable MSI oscillator.
Return values	• <b>None:</b>
Reference Manual to LL API cross reference:	• CR MSION LL_RCC_MSI_Disable

### LL\_RCC\_MSI\_IsReady

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_MSI_IsReady (void)</code>
Function Description	Check if MSI oscillator Ready.
Return values	• <b>State:</b> of bit (1 or 0).
Reference Manual to LL API cross reference:	• CR MSIRDY LL_RCC_MSI_IsReady

### LL\_RCC\_MSI\_SetRange

Function Name	<code>__STATIC_INLINE void LL_RCC_MSI_SetRange (uint32_t Range)</code>
Function Description	Configure the Internal Multi Speed oscillator (MSI) clock range in run mode.
Parameters	• <b>Range:</b> This parameter can be one of the following values:

- LL\_RCC\_MSIRANGE\_0
- LL\_RCC\_MSIRANGE\_1
- LL\_RCC\_MSIRANGE\_2
- LL\_RCC\_MSIRANGE\_3
- LL\_RCC\_MSIRANGE\_4
- LL\_RCC\_MSIRANGE\_5
- LL\_RCC\_MSIRANGE\_6

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- ICSCR MSIRANGE LL\_RCC\_MSI\_SetRange

### **LL\_RCC\_MSI\_GetRange**

Function Name **\_STATIC\_INLINE uint32\_t LL\_RCC\_MSI\_GetRange (void )**

Function Description Get the Internal Multi Speed oscillator (MSI) clock range in run mode.

Return values

- **Returned:** value can be one of the following values:

- LL\_RCC\_MSIRANGE\_0
- LL\_RCC\_MSIRANGE\_1
- LL\_RCC\_MSIRANGE\_2
- LL\_RCC\_MSIRANGE\_3
- LL\_RCC\_MSIRANGE\_4
- LL\_RCC\_MSIRANGE\_5
- LL\_RCC\_MSIRANGE\_6

Reference Manual to  
LL API cross  
reference:

- ICSCR MSIRANGE LL\_RCC\_MSI\_GetRange

### **LL\_RCC\_MSI\_GetCalibration**

Function Name **\_STATIC\_INLINE uint32\_t LL\_RCC\_MSI\_GetCalibration (void )**

Function Description Get MSI Calibration value.

Return values

- **Between:** Min\_Data = 0x00 and Max\_Data = 0xFF

Notes

- When MSITRIM is written, MSICAL is updated with the sum of MSITRIM and the factory trim value

Reference Manual to  
LL API cross  
reference:

- ICSCR MSICAL LL\_RCC\_MSI\_GetCalibration

### **LL\_RCC\_MSI\_SetCalibTrimming**

Function Name **\_STATIC\_INLINE void LL\_RCC\_MSI\_SetCalibTrimming (uint32\_t Value)**

Function Description Set MSI Calibration trimming.

Parameters

- **Value:** between Min\_Data = 0x00 and Max\_Data = 0xFF

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>user-programmable trimming value that is added to the MSICAL</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ICSCR MSITRIM LL_RCC_MSI_SetCalibTrimming</li> </ul>

### **LL\_RCC\_MSI\_SetCalibTrimming**

Function Name	<b><u>__STATIC_INLINE uint32_t LL_RCC_MSI_SetCalibTrimming(void )</u></b>
Function Description	Get MSI Calibration trimming.
Return values	<ul style="list-style-type: none"> <li><b>Between:</b> Min_Data = 0x00 and Max_Data = 0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ICSCR MSITRIM LL_RCC_MSI_SetCalibTrimming</li> </ul>

### **LL\_RCC\_SetSysClkSource**

Function Name	<b><u>__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)</u></b>
Function Description	Configure the system clock source.
Parameters	<ul style="list-style-type: none"> <li><b>Source:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_SYS_CLKSOURCE_MSI</li> <li>- LL_RCC_SYS_CLKSOURCE_HSI</li> <li>- LL_RCC_SYS_CLKSOURCE_HSE</li> <li>- LL_RCC_SYS_CLKSOURCE_PLL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR SW LL_RCC_SetSysClkSource</li> </ul>

### **LL\_RCC\_GetSysClkSource**

Function Name	<b><u>__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void )</u></b>
Function Description	Get the system clock source.
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_SYS_CLKSOURCE_STATUS_MSI</li> <li>- LL_RCC_SYS_CLKSOURCE_STATUS_HSI</li> <li>- LL_RCC_SYS_CLKSOURCE_STATUS_HSE</li> <li>- LL_RCC_SYS_CLKSOURCE_STATUS_PLL</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CFGR SWS LL_RCC_GetSysClkSource</li> </ul>

**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  
reference:  
• CFGR PPRE1 LL\_RCC\_SetAPB1Prescaler

**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\_PPREG 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  
LL API cross  
reference:

- CFGR\_PPREG1 LL\_RCC\_GetAPB1Prescaler

### **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\_PPREG2 LL\_RCC\_GetAPB2Prescaler

### **LL\_RCC\_SetClkAfterWakeFromStop**

Function Name	<b><code>_STATIC_INLINE void LL_RCC_SetClkAfterWakeFromStop (uint32_t Clock)</code></b>
Function Description	Set Clock After Wake-Up From Stop mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Clock:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_STOP_WAKEUPCLOCK_MSI</li> <li>- LL_RCC_STOP_WAKEUPCLOCK_HSI</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR_STOPWUCK LL_RCC_SetClkAfterWakeFromStop</li> </ul>

### **LL\_RCC\_GetClkAfterWakeFromStop**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RCC_GetClkAfterWakeFromStop (void )</code></b>
Function Description	Get Clock After Wake-Up From Stop mode.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_STOP_WAKEUPCLOCK_MSI</li> <li>- LL_RCC_STOP_WAKEUPCLOCK_HSI</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFGR_STOPWUCK LL_RCC_GetClkAfterWakeFromStop</li> </ul>

### **LL\_RCC\_ConfigMCO**

Function Name	<b><code>_STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)</code></b>
Function Description	Configure MCOx.
Parameters	<ul style="list-style-type: none"> <li>• <b>MCOxSource:</b> This parameter can be one of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>- LL_RCC_MCO1SOURCE_NOCLOCK</li> <li>- LL_RCC_MCO1SOURCE_SYSCLK</li> <li>- LL_RCC_MCO1SOURCE_HSI</li> <li>- LL_RCC_MCO1SOURCE_MSI</li> <li>- LL_RCC_MCO1SOURCE_HSE</li> <li>- LL_RCC_MCO1SOURCE_PLLCLK</li> <li>- LL_RCC_MCO1SOURCE_LSI</li> <li>- LL_RCC_MCO1SOURCE_LSE</li> <li>- LL_RCC_MCO1SOURCE_HSI48 (*)</li> </ul> </li> <li>• <b>MCOxPrescaler:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_MCO1_DIV_1</li> <li>- LL_RCC_MCO1_DIV_2</li> </ul> </li> </ul>

- LL\_RCC\_MCO1\_DIV\_4
- LL\_RCC\_MCO1\_DIV\_8
- LL\_RCC\_MCO1\_DIV\_16

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- CFGR MCSEL LL\_RCC\_ConfigMCO
- CFGR MCOPRE LL\_RCC\_ConfigMCO

### **LL\_RCC\_SetUSARTClockSource**

Function Name      **\_\_STATIC\_INLINE void LL\_RCC\_SetUSARTClockSource (uint32\_t USARTxSource)**

Function Description      Configure USARTx clock source.

Parameters

- **USARTxSource:** This parameter can be one of the following values: (\*) value not defined in all devices.
  - LL\_RCC\_USART1\_CLKSOURCE\_PCLK2 (\*)
  - LL\_RCC\_USART1\_CLKSOURCE\_SYSCLK (\*)
  - LL\_RCC\_USART1\_CLKSOURCE\_HSI (\*)
  - LL\_RCC\_USART1\_CLKSOURCE\_LSE (\*)
  - LL\_RCC\_USART2\_CLKSOURCE\_PCLK1
  - LL\_RCC\_USART2\_CLKSOURCE\_SYSCLK
  - LL\_RCC\_USART2\_CLKSOURCE\_HSI
  - LL\_RCC\_USART2\_CLKSOURCE\_LSE

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- CCIPR USARTxSEL LL\_RCC\_SetUSARTClockSource

### **LL\_RCC\_SetLPUARTClockSource**

Function Name      **\_\_STATIC\_INLINE void LL\_RCC\_SetLPUARTClockSource (uint32\_t LPUARTxSource)**

Function Description      Configure LPUART1x clock source.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL\_RCC\_LPUART1\_CLKSOURCE\_PCLK1
  - LL\_RCC\_LPUART1\_CLKSOURCE\_SYSCLK
  - LL\_RCC\_LPUART1\_CLKSOURCE\_HSI
  - LL\_RCC\_LPUART1\_CLKSOURCE\_LSE

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- CCIPR LPUART1SEL LL\_RCC\_SetLPUARTClockSource

### **LL\_RCC\_SetI2CCClockSource**

Function Name      **\_\_STATIC\_INLINE void LL\_RCC\_SetI2CCClockSource (uint32\_t I2CxSource)**

Function Description	Configure I2Cx clock source.
Parameters	<ul style="list-style-type: none"> <li><b>I2CxSource:</b> This parameter can be one of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RCC_I2C1_CLKSOURCE_PCLK1</li> <li>- LL_RCC_I2C1_CLKSOURCE_SYSCLK</li> <li>- LL_RCC_I2C1_CLKSOURCE_HSI</li> <li>- LL_RCC_I2C3_CLKSOURCE_PCLK1 (*)</li> <li>- LL_RCC_I2C3_CLKSOURCE_SYSCLK (*)</li> <li>- LL_RCC_I2C3_CLKSOURCE_HSI (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCIPR I2CxSEL LL_RCC_SetI2CClockSource</li> </ul>

### LL\_RCC\_SetLPTIMClockSource

Function Name	<code>__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t LPTIMxSource)</code>
Function Description	Configure LPTIMx clock source.
Parameters	<ul style="list-style-type: none"> <li><b>LPTIMxSource:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RCC_LPTIM1_CLKSOURCE_PCLK1</li> <li>- LL_RCC_LPTIM1_CLKSOURCE_LSI</li> <li>- LL_RCC_LPTIM1_CLKSOURCE_HSI</li> <li>- LL_RCC_LPTIM1_CLKSOURCE_LSE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCIPR LPTIMxSEL LL_RCC_SetLPTIMClockSource</li> </ul>

### LL\_RCC\_SetRNGClockSource

Function Name	<code>__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t RNGxSource)</code>
Function Description	Configure RNG clock source.
Parameters	<ul style="list-style-type: none"> <li><b>RNGxSource:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RCC_RNG_CLKSOURCE_PLL</li> <li>- LL_RCC_RNG_CLKSOURCE_HSI48</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCIPR HSI48SEL LL_RCC_SetRNGClockSource</li> </ul>

### LL\_RCC\_SetUSBClockSource

Function Name	<code>__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)</code>
---------------	--

Function Description	Configure USB clock source.
Parameters	<ul style="list-style-type: none"> <li><b>USBxSource:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC RNG_CLKSOURCE_PLL</li> <li>- LL_RCC RNG_CLKSOURCE_HSI48</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCIPR HSI48SEL LL_RCC_SetUSBClockSource</li> </ul>

**LL\_RCC\_GetUSARTClockSource**

Function Name	<b>_STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTx)</b>
Function Description	Get USARTx clock source.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_USART1_CLKSOURCE (*)</li> <li>- LL_RCC_USART2_CLKSOURCE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>- LL_RCC_USART1_CLKSOURCE_PCLK2 (*)</li> <li>- LL_RCC_USART1_CLKSOURCE_SYSCLK (*)</li> <li>- LL_RCC_USART1_CLKSOURCE_HSI (*)</li> <li>- LL_RCC_USART1_CLKSOURCE_LSE (*)</li> <li>- LL_RCC_USART2_CLKSOURCE_PCLK1</li> <li>- LL_RCC_USART2_CLKSOURCE_SYSCLK</li> <li>- LL_RCC_USART2_CLKSOURCE_HSI</li> <li>- LL_RCC_USART2_CLKSOURCE_LSE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCIPR USARTxSEL LL_RCC_GetUSARTClockSource</li> </ul>

**LL\_RCC\_GetLPUARTClockSource**

Function Name	<b>_STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t LPUARTx)</b>
Function Description	Get LPUARTx clock source.
Parameters	<ul style="list-style-type: none"> <li><b>LPUARTx:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_LPUART1_CLKSOURCE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_LPUART1_CLKSOURCE_PCLK1</li> <li>- LL_RCC_LPUART1_CLKSOURCE_SYSCLK</li> <li>- LL_RCC_LPUART1_CLKSOURCE_HSI</li> <li>- LL_RCC_LPUART1_CLKSOURCE_LSE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCIPR LPUART1SEL LL_RCC_GetLPUARTClockSource</li> </ul>

**LL\_RCC\_GetI2CClockSource**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)</code></b>
Function Description	Get I2Cx clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2Cx:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_I2C1_CLKSOURCE</li> <li>- LL_RCC_I2C3_CLKSOURCE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>- LL_RCC_I2C1_CLKSOURCE_PCLK1</li> <li>- LL_RCC_I2C1_CLKSOURCE_SYSCLK</li> <li>- LL_RCC_I2C1_CLKSOURCE_HSI</li> <li>- LL_RCC_I2C3_CLKSOURCE_PCLK1 (*)</li> <li>- LL_RCC_I2C3_CLKSOURCE_SYSCLK (*)</li> <li>- LL_RCC_I2C3_CLKSOURCE_HSI (*)</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCIPR I2CxSEL LL_RCC_GetI2CClockSource</li> </ul>

**LL\_RCC\_GetLPTIMClockSource**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t LPTIMx)</code></b>
Function Description	Get LPTIMx clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMx:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_LPTIM1_CLKSOURCE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_LPTIM1_CLKSOURCE_PCLK1</li> <li>- LL_RCC_LPTIM1_CLKSOURCE_LSI</li> <li>- LL_RCC_LPTIM1_CLKSOURCE_HSI</li> <li>- LL_RCC_LPTIM1_CLKSOURCE_LSE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCIPR LPTIMxSEL LL_RCC_GetLPTIMClockSource</li> </ul>

**LL\_RCC\_GetRNGClockSource**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t RNGx)</code></b>
Function Description	Get RNGx clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>RNGx:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_RNG_CLKSOURCE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_RNG_CLKSOURCE_PLL</li> <li>- LL_RCC_RNG_CLKSOURCE_HSI48</li> </ul> </li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CCIPR CLK48SEL LL_RCC_GetRNGClockSource</li> </ul>

reference:

### **LL\_RCC\_GetUSBClockSource**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource(uint32_t USBx)</code></b>
Function Description	Get USBx clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>USBx:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_RCC_USB_CLKSOURCE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_RCC_USB_CLKSOURCE_PLL</code></li> <li>- <code>LL_RCC_USB_CLKSOURCE_HSI48</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCIPR CLK48SEL <code>LL_RCC_GetUSBClockSource</code></li> </ul>

### **LL\_RCC\_SetRTCClockSource**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_SetRTCClockSource(uint32_t Source)</code></b>
Function Description	Set RTC Clock Source.
Parameters	<ul style="list-style-type: none"> <li>• <b>Source:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_RCC_RTC_CLKSOURCE_NONE</code></li> <li>- <code>LL_RCC_RTC_CLKSOURCE_LSE</code></li> <li>- <code>LL_RCC_RTC_CLKSOURCE_LSI</code></li> <li>- <code>LL_RCC_RTC_CLKSOURCE_HSE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Once the RTC clock source has been selected, it cannot be changed any more unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The RTCRST bit can be used to reset them.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CSR RTCSEL <code>LL_RCC_SetRTCClockSource</code></li> </ul>

### **LL\_RCC\_GetRTCClockSource**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource(void)</code></b>
Function Description	Get RTC Clock Source.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_RCC_RTC_CLKSOURCE_NONE</code></li> <li>- <code>LL_RCC_RTC_CLKSOURCE_LSE</code></li> <li>- <code>LL_RCC_RTC_CLKSOURCE_LSI</code></li> <li>- <code>LL_RCC_RTC_CLKSOURCE_HSE</code></li> </ul> </li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CSR RTCSEL <code>LL_RCC_GetRTCClockSource</code></li> </ul>

reference:

### **LL\_RCC\_EnableRTC**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableRTC (void )</code></b>
Function Description	Enable RTC.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CSR RTCEN LL_RCC_EnableRTC</li></ul>

### **LL\_RCC\_DisableRTC**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableRTC (void )</code></b>
Function Description	Disable RTC.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CSR RTCEN LL_RCC_DisableRTC</li></ul>

### **LL\_RCC\_IsEnabledRTC**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void )</code></b>
Function Description	Check if RTC has been enabled or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CSR RTCEN LL_RCC_IsEnabledRTC</li></ul>

### **LL\_RCC\_ForceBackupDomainReset**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void )</code></b>
Function Description	Force the Backup domain reset.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CSR RTCRST LL_RCC_ForceBackupDomainReset</li></ul>

### **LL\_RCC\_ReleaseBackupDomainReset**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void )</code></b>
Function Description	Release the Backup domain reset.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to	<ul style="list-style-type: none"><li>• CSR RTCRST LL_RCC_ReleaseBackupDomainReset</li></ul>

LL API cross  
reference:

### 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 CR PLLON LL\_RCC\_PLL\_Disable  
LL API cross  
reference:

### 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 CR PLLRDY LL\_RCC\_PLL\_IsReady  
LL API cross  
reference:

### LL\_RCC\_PLL\_ConfigDomain\_SYS

Function Name **`__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SYS  
(uint32_t Source, uint32_t PLLMul, uint32_t PLLDiv)`**

Function Description Configure PLL used for SYSCLK Domain.

Parameters

- **Source:** This parameter can be one of the following values:
  - `LL_RCC_PLLSOURCE_HSI`
  - `LL_RCC_PLLSOURCE_HSE`
- **PLLMul:** This parameter can be one of the following values:
  - `LL_RCC_PLL_MUL_3`
  - `LL_RCC_PLL_MUL_4`
  - `LL_RCC_PLL_MUL_6`
  - `LL_RCC_PLL_MUL_8`
  - `LL_RCC_PLL_MUL_12`
  - `LL_RCC_PLL_MUL_16`
  - `LL_RCC_PLL_MUL_24`

- LL\_RCC\_PLL\_MUL\_32
  - LL\_RCC\_PLL\_MUL\_48
  - **PLLDiv:** This parameter can be one of the following values:
    - LL\_RCC\_PLL\_DIV\_2
    - LL\_RCC\_PLL\_DIV\_3
    - LL\_RCC\_PLL\_DIV\_4
  - **None:**
- Return values
- Reference Manual to LL API cross reference:
- CFGR PLLSRC LL\_RCC\_PLL\_ConfigDomain\_SYS
  - CFGR PLLMUL LL\_RCC\_PLL\_ConfigDomain\_SYS
  - CFGR PLLDIV LL\_RCC\_PLL\_ConfigDomain\_SYS

### **LL\_RCC\_PLL\_GetMainSource**

- |   |  |
|---|--|
| Function Name                               | <b><code>__STATIC_INLINE uint32_t LL_RCC_PLL_GetMainSource(void)</code></b>  |
| Function Description                        | Get the oscillator used as PLL clock source.   |
| Return values                               | <ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_PLLSOURCE_HSI</li> <li>- LL_RCC_PLLSOURCE_HSE</li> </ul> </li> </ul> |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none"> <li>• CFGR PLLSRC LL_RCC_PLL_GetMainSource</li> </ul>   |

### **LL\_RCC\_PLL\_GetMultiplicator**

- |   |   |
|---|---|
| Function Name                               | <b><code>__STATIC_INLINE uint32_t LL_RCC_PLL_GetMultiplicator(void)</code></b>  |
| Function Description                        | Get PLL multiplication Factor.  |
| Return values                               | <ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_PLL_MUL_3</li> <li>- LL_RCC_PLL_MUL_4</li> <li>- LL_RCC_PLL_MUL_6</li> <li>- LL_RCC_PLL_MUL_8</li> <li>- LL_RCC_PLL_MUL_12</li> <li>- LL_RCC_PLL_MUL_16</li> <li>- LL_RCC_PLL_MUL_24</li> <li>- LL_RCC_PLL_MUL_32</li> <li>- LL_RCC_PLL_MUL_48</li> </ul> </li> </ul> |
| Reference Manual to LL API cross reference: | <ul style="list-style-type: none"> <li>• CFGR PLLMUL LL_RCC_PLL_GetMultiplicator</li> </ul>   |

### **LL\_RCC\_PLL\_GetDivider**

- |                      |  |
|----------------------|--|
| Function Name        | <b><code>__STATIC_INLINE uint32_t LL_RCC_PLL_GetDivider(void)</code></b>   |
| Function Description | Get Division factor for the main PLL and other PLL.  |
| Return values        | <ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RCC_PLL_DIV_2</li> </ul> </li> </ul> |

- LL\_RCC\_PLL\_DIV\_3
- LL\_RCC\_PLL\_DIV\_4

Reference Manual to  
LL API cross  
reference:

- CFGR PLLDIV LL\_RCC\_PLL\_GetDivider

### 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:  
CICR 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:  
CICR LSERDYC LL\_RCC\_ClearFlag\_LSERDY

### LL\_RCC\_ClearFlag\_MSIRDY

Function Name **\_\_STATIC\_INLINE void LL\_RCC\_ClearFlag\_MSIRDY (void )**

Function Description Clear MSI ready interrupt flag.

Return values • **None:**

Reference Manual to  
LL API cross  
reference:  
CICR MSIRDYC LL\_RCC\_ClearFlag\_MSIRDY

### 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:  
CICR 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	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CICR HSERDYC LL_RCC_ClearFlag_HSERDY</li> </ul>

### **LL\_RCC\_ClearFlag\_PLLRDY**

Function Name	<code>__STATIC_INLINE void LL_RCC_ClearFlag_PLLRDY (void )</code>
Function Description	Clear PLL ready interrupt flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CICR PLLRDYC LL_RCC_ClearFlag_PLLRDY</li> </ul>

### **LL\_RCC\_ClearFlag\_HSI48RDY**

Function Name	<code>__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void )</code>
Function Description	Clear HSI48 ready interrupt flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY</li> </ul>

### **LL\_RCC\_ClearFlag\_HSECSS**

Function Name	<code>__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void )</code>
Function Description	Clear Clock security system interrupt flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CICR CSSC LL_RCC_ClearFlag_HSECSS</li> </ul>

### **LL\_RCC\_ClearFlag\_LSECSS**

Function Name	<code>__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void )</code>
Function Description	Clear LSE Clock security system interrupt flag.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CICR LSECSSC LL_RCC_ClearFlag_LSECSS</li> </ul>

### **LL\_RCC\_IsActiveFlag\_LSIRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void )</code>
Function Description	Check if LSI ready interrupt occurred or not.

---

Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY</li></ul>

### LL\_RCC\_IsActiveFlag\_LSERDY

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY(void)</code>
Function Description	Check if LSE ready interrupt occurred or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY</li></ul>

### LL\_RCC\_IsActiveFlag\_MSIRDY

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_MSIRDY(void)</code>
Function Description	Check if MSI ready interrupt occurred or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIFR MSIRDYF LL_RCC_IsActiveFlag_MSIRDY</li></ul>

### LL\_RCC\_IsActiveFlag\_HSIRDY

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY(void)</code>
Function Description	Check if HSI ready interrupt occurred or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY</li></ul>

### LL\_RCC\_IsActiveFlag\_HSERDY

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY(void)</code>
Function Description	Check if HSE ready interrupt occurred or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY</li></ul>

**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:  
• CIRF PLLRDYF LL\_RCC\_IsActiveFlag\_PLLRDY

**LL\_RCC\_IsActiveFlag\_HSI48RDY**

Function Name      `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY(void)`

Function Description      Check if HSI48 ready interrupt occurred or not.

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
• CIRF HSI48RDYF LL\_RCC\_IsActiveFlag\_HSI48RDY

**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:  
• CIRF CSSF LL\_RCC\_IsActiveFlag\_HSECSS

**LL\_RCC\_IsActiveFlag\_LSECSS**

Function Name      `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS(void)`

Function Description      Check if LSE Clock security system interrupt occurred or not.

Return values      • **State:** of bit (1 or 0).

Reference Manual to  
LL API cross  
reference:  
• CIRF LSECSSF LL\_RCC\_IsActiveFlag\_LSECSS

**LL\_RCC\_IsActiveFlag\_HSIDIV**

Function Name      `__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIDIV(void)`

Function Description      Check if HSI Divider is enabled (it divides by 4)

Return values      • **State:** of bit (1 or 0).

- Reference Manual to  
LL API cross  
reference:
- CR HSIDIVF LL\_RCC\_IsActiveFlag\_HSIDIV

### LL\_RCC\_IsActiveFlag\_FWRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_FWRST(void)</code>
Function Description	Check if RCC flag FW reset is set or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• CSR FWRSTF LL_RCC_IsActiveFlag_FWRST

### LL\_RCC\_IsActiveFlag\_IWDGRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST(void)</code>
Function Description	Check if RCC flag Independent Watchdog reset is set or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

### LL\_RCC\_IsActiveFlag\_LPWRRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST(void)</code>
Function Description	Check if RCC flag Low Power reset is set or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

### LL\_RCC\_IsActiveFlag\_OBLRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_OBLRST(void)</code>
Function Description	Check if RCC flag is set or not.
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	• CSR OBLRSTF LL_RCC_IsActiveFlag_OBLRST

### LL\_RCC\_IsActiveFlag\_PINRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST(void)</code>
---------------	--

Function Description	Check if RCC flag Pin reset is set or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

### LL\_RCC\_IsActiveFlag\_PORRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST( void )</code>
Function Description	Check if RCC flag POR/PDR reset is set or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CSR PORRSTF LL_RCC_IsActiveFlag_PORRST

### LL\_RCC\_IsActiveFlag\_SFTRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST( void )</code>
Function Description	Check if RCC flag Software reset is set or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

### LL\_RCC\_IsActiveFlag\_WWDGRST

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST( void )</code>
Function Description	Check if RCC flag Window Watchdog reset is set or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

### LL\_RCC\_ClearResetFlags

Function Name	<code>__STATIC_INLINE void LL_RCC_ClearResetFlags (void )</code>
Function Description	Set RMVF bit to clear the reset flags.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	CSR RMVF LL_RCC_ClearResetFlags

**LL\_RCC\_EnableIT\_LSIRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)</code></b>
Function Description	Enable LSI ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY</li></ul>

**LL\_RCC\_EnableIT\_LSERDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)</code></b>
Function Description	Enable LSE ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER LSERDYIE LL_RCC_EnableIT_LSERDY</li></ul>

**LL\_RCC\_EnableIT\_MSIRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_MSIRDY (void)</code></b>
Function Description	Enable MSI ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER MSIRDYIE LL_RCC_EnableIT_MSIRDY</li></ul>

**LL\_RCC\_EnableIT\_HSIRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)</code></b>
Function Description	Enable HSI ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY</li></ul>

**LL\_RCC\_EnableIT\_HSERDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)</code></b>
Function Description	Enable HSE ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER HSERDYIE LL_RCC_EnableIT_HSERDY</li></ul>

**LL\_RCC\_EnableIT\_PLLRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_PLLRDY (void )</code></b>
Function Description	Enable PLL ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER PLLRDYIE LL_RCC_EnableIT_PLLRDY</li></ul>

**LL\_RCC\_EnableIT\_HSI48RDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void )</code></b>
Function Description	Enable HSI48 ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY</li></ul>

**LL\_RCC\_EnableIT\_LSECSS**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_EnableIT_LSECSS (void )</code></b>
Function Description	Enable LSE clock security system interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER LSECSSIE LL_RCC_EnableIT_LSECSS</li></ul>

**LL\_RCC\_DisableIT\_LSIRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void )</code></b>
Function Description	Disable LSI ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY</li></ul>

**LL\_RCC\_DisableIT\_LSERDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void )</code></b>
Function Description	Disable LSE ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER LSERDYIE LL_RCC_DisableIT_LSERDY</li></ul>

**LL\_RCC\_DisableIT\_MSIRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_MSIRDY (void )</code></b>
Function Description	Disable MSI ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER MSIRDYIE LL_RCC_DisableIT_MSIRDY</li></ul>

**LL\_RCC\_DisableIT\_HSIRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void )</code></b>
Function Description	Disable HSI ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY</li></ul>

**LL\_RCC\_DisableIT\_HSERDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void )</code></b>
Function Description	Disable HSE ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER HSERDYIE LL_RCC_DisableIT_HSERDY</li></ul>

**LL\_RCC\_DisableIT\_PLLRDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_PLLRDY (void )</code></b>
Function Description	Disable PLL ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER PLLRDYIE LL_RCC_DisableIT_PLLRDY</li></ul>

**LL\_RCC\_DisableIT\_HSI48RDY**

Function Name	<b><code>__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void )</code></b>
Function Description	Disable HSI48 ready interrupt.
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY</li></ul>

**LL\_RCC\_DisableIT\_LSECSS**

Function Name	<code>__STATIC_INLINE void LL_RCC_DisableIT_LSECSS (void )</code>
Function Description	Disable LSE clock security system interrupt.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CIER LSECSSIE LL_RCC_DisableIT_LSECSS</li> </ul>

**LL\_RCC\_IsEnabledIT\_LSIRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void )</code>
Function Description	Checks if LSI ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY</li> </ul>

**LL\_RCC\_IsEnabledIT\_LSERDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void )</code>
Function Description	Checks if LSE ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY</li> </ul>

**LL\_RCC\_IsEnabledIT\_MSIRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_MSIRDY (void )</code>
Function Description	Checks if MSI ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CIER MSIRDYIE LL_RCC_IsEnabledIT_MSIRDY</li> </ul>

**LL\_RCC\_IsEnabledIT\_HSIRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void )</code>
Function Description	Checks if HSI ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY</li> </ul>

reference:

### **LL\_RCC\_IsEnabledIT\_HSERDY**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY(void)</code></b>
Function Description	Checks if HSE ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

### **LL\_RCC\_IsEnabledIT\_PLLRDY**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLRDY(void)</code></b>
Function Description	Checks if PLL ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CIER PLLRDYIE LL_RCC_IsEnabledIT_PLLRDY

### **LL\_RCC\_IsEnabledIT\_HSI48RDY**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY(void)</code></b>
Function Description	Checks if HSI48 ready interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CIER HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

### **LL\_RCC\_IsEnabledIT\_LSECSS**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSECSS(void)</code></b>
Function Description	Checks if LSECSS interrupt source is enabled or disabled.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	• CIER LSECSSIE LL_RCC_IsEnabledIT_LSECSS

### **LL\_RCC\_DelInit**

Function Name	<b><code>ErrorStatus LL_RCC_DelInit(void)</code></b>
Function Description	Reset the RCC clock configuration to the default reset state.
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- SUCCESS: RCC registers are de-initialized</li> <li>- ERROR: not applicable</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The default reset state of the clock configuration is given below: MSI ON and used as system clock sourceHSE, HSI and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFFAll interrupts disabled</li> <li>• This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks</li> </ul>

### LL\_RCC\_GetSystemClocksFreq

Function Name	<b>void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)</b>
Function Description	Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_Clocks:</b> pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

### LL\_RCC\_GetUSARTClockFreq

Function Name	<b>uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)</b>
Function Description	Return USARTx clock frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTxSource:</b> This parameter can be one of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RCC_USART1_CLKSOURCE</li> <li>- LL_RCC_USART2_CLKSOURCE (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>USART:</b> clock frequency (in Hz) <ul style="list-style-type: none"> <li>- LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready</li> </ul> </li> </ul>

### LL\_RCC\_GetI2CClockFreq

Function Name	<b>uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)</b>
Function Description	Return I2Cx clock frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>I2CxSource:</b> This parameter can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_RCC_I2C1_CLKSOURCE</li> <li>- LL_RCC_I2C3_CLKSOURCE (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>I2C:</b> clock frequency (in Hz) <ul style="list-style-type: none"> <li>- LL_RCC_PERIPH_FREQUENCY_NO indicates that HSI oscillator is not ready</li> </ul> </li> </ul>

**LL\_RCC\_GetLPUARTClockFreq**

Function Name	<code>uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)</code>
Function Description	Return LPUARTx clock frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPUARTxSource:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <code>LL_RCC_LPUART1_CLKSOURCE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>LPUART:</b> clock frequency (in Hz)           <ul style="list-style-type: none"> <li>– <code>LL_RCC_PERIPH_FREQUENCY_NO</code> indicates that oscillator (HSI or LSE) is not ready</li> </ul> </li> </ul>

**LL\_RCC\_GetLPTIMClockFreq**

Function Name	<code>uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)</code>
Function Description	Return LPTIMx clock frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>LPTIMxSource:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <code>LL_RCC_LPTIM1_CLKSOURCE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>LPTIM:</b> clock frequency (in Hz)           <ul style="list-style-type: none"> <li>– <code>LL_RCC_PERIPH_FREQUENCY_NO</code> indicates that oscillator (HSI or LSE) is not ready</li> </ul> </li> </ul>

**LL\_RCC\_GetUSBClockFreq**

Function Name	<code>uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)</code>
Function Description	Return USBx clock frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>USBxSource:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <code>LL_RCC_USB_CLKSOURCE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>USB:</b> clock frequency (in Hz)           <ul style="list-style-type: none"> <li>– <code>LL_RCC_PERIPH_FREQUENCY_NO</code> indicates that oscillator (HSI48) or PLL is not ready</li> <li>– <code>LL_RCC_PERIPH_FREQUENCY_NA</code> indicates that no clock source selected</li> </ul> </li> </ul>

## 69.3 RCC Firmware driver defines

### 69.3.1 RCC

#### *APB low-speed prescaler (APB1)*

<code>LL_RCC_APB1_DIV_1</code>	HCLK not divided
<code>LL_RCC_APB1_DIV_2</code>	HCLK divided by 2
<code>LL_RCC_APB1_DIV_4</code>	HCLK divided by 4
<code>LL_RCC_APB1_DIV_8</code>	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_CICR_LSIRDYC` LSI Ready Interrupt Clear

`LL_RCC_CICR_LSERDYC` LSE Ready Interrupt Clear

`LL_RCC_CICR_HSIRDYC` HSI Ready Interrupt Clear

`LL_RCC_CICR_HSERDYC` HSE Ready Interrupt Clear

`LL_RCC_CICR_PLLRDYC` PLL Ready Interrupt Clear

`LL_RCC_CICR_MSIRDYC` MSI Ready Interrupt Clear

`LL_RCC_CICR_HSI48RDYC` HSI48 Ready Interrupt Clear

`LL_RCC_CICR_LSECSSC` LSE Clock Security System Interrupt Clear

`LL_RCC_CICR_CSSC` Clock Security System Interrupt Clear

***Get Flags Defines***

`LL_RCC_CIFR_LSIRDYF` LSI Ready Interrupt flag

`LL_RCC_CIFR_LSERDYF` LSE Ready Interrupt flag

`LL_RCC_CIFR_HSIRDYF` HSI Ready Interrupt flag

`LL_RCC_CIFR_HSERDYF` HSE Ready Interrupt flag

`LL_RCC_CIFR_PLLRDYF` PLL Ready Interrupt flag

`LL_RCC_CIFR_MSIRDYF` MSI Ready Interrupt flag

`LL_RCC_CIFR_HSI48RDYF` HSI48 Ready Interrupt flag

`LL_RCC_CIFR_LSECSSF` LSE Clock Security System Interrupt flag

`LL_RCC_CIFR_CSSF` Clock Security System Interrupt flag

`LL_RCC_CSR_FWRSTF` Firewall reset flag

`LL_RCC_CSR_OBLRSTF` OBL reset 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

***Peripheral I2C get clock source***

`LL_RCC_I2C1_CLKSOURCE` I2C1 clock source selection bits  
`LL_RCC_I2C3_CLKSOURCE` I2C3 clock source selection bits

***Peripheral I2C clock source selection***

`LL_RCC_I2C1_CLKSOURCE_PCLK1` PCLK1 selected as I2C1 clock  
`LL_RCC_I2C1_CLKSOURCE_SYSCLK` SYSCLK selected as I2C1 clock  
`LL_RCC_I2C1_CLKSOURCE_HSI` HSI selected as I2C1 clock  
`LL_RCC_I2C3_CLKSOURCE_PCLK1` PCLK1 selected as I2C3 clock  
`LL_RCC_I2C3_CLKSOURCE_SYSCLK` SYSCLK selected as I2C3 clock  
`LL_RCC_I2C3_CLKSOURCE_HSI` HSI selected as I2C3 clock

***IT Defines***

`LL_RCC_CIER_LSIRDYIE` LSI Ready Interrupt Enable  
`LL_RCC_CIER_LSERDYIE` LSE Ready Interrupt Enable  
`LL_RCC_CIER_HSIRDYIE` HSI Ready Interrupt Enable  
`LL_RCC_CIER_HSERDYIE` HSE Ready Interrupt Enable  
`LL_RCC_CIER_PLLRDYIE` PLL Ready Interrupt Enable  
`LL_RCC_CIER_MSIRDYIE` MSI Ready Interrupt Enable  
`LL_RCC_CIER_HSI48RDYIE` HSI48 Ready Interrupt Enable  
`LL_RCC_CIER_LSECSSIE` LSE CSS Interrupt Enable

***Peripheral LPTIM get clock source***

`LL_RCC_LPTIM1_CLKSOURCE` LPTIM1 clock source selection bits

***Peripheral LPTIM clock source selection***

`LL_RCC_LPTIM1_CLKSOURCE_PCLK1` PCLK1 selected as LPTIM1 clock  
`LL_RCC_LPTIM1_CLKSOURCE_LSI` LSI selected as LPTIM1 clock  
`LL_RCC_LPTIM1_CLKSOURCE_HSI` HSI selected as LPTIM1 clock  
`LL_RCC_LPTIM1_CLKSOURCE_LSE` LSE selected as LPTIM1 clock

***Peripheral LPUART get clock source***

`LL_RCC_LPUART1_CLKSOURCE` LPUART1 clock source selection bits

***Peripheral LPUART clock source selection***

`LL_RCC_LPUART1_CLKSOURCE_PCLK1` PCLK1 selected as LPUART1 clock  
`LL_RCC_LPUART1_CLKSOURCE_SYSCLK` SYSCLK selected as LPUART1 clock  
`LL_RCC_LPUART1_CLKSOURCE_HSI` HSI selected as LPUART1 clock  
`LL_RCC_LPUART1_CLKSOURCE_LSE` LSE selected as LPUART1 clock

***LSE oscillator drive capability***

`LL_RCC_LSEDRIVE_LOW` Xtal mode lower driving capability  
`LL_RCC_LSEDRIVE_MEDIUMLOW` Xtal mode medium low driving capability  
`LL_RCC_LSEDRIVE_MEDIUMHIGH` Xtal mode medium high driving capability

LL_RCC_LSEDRIVE_HIGH	Xtal mode higher driving capability
<b>MCO1 SOURCE selection</b>	
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_MS1	MSI selection as MCO source
LL_RCC_MCO1SOURCE_HSE	HSE selection as MCO source
LL_RCC_MCO1SOURCE_LSI	LSI selection as MCO source
LL_RCC_MCO1SOURCE_LSE	LSE selection as MCO source
LL_RCC_MCO1SOURCE_HSI48	HSI48 selection as MCO source
LL_RCC_MCO1SOURCE_PLLCLK	PLLCLK selection as MCO source
<b>MCO1 prescaler</b>	
LL_RCC_MCO1_DIV_1	MCO Clock divided by 1
LL_RCC_MCO1_DIV_2	MCO Clock divided by 2
LL_RCC_MCO1_DIV_4	MCO Clock divided by 4
LL_RCC_MCO1_DIV_8	MCO Clock divided by 8
LL_RCC_MCO1_DIV_16	MCO Clock divided by 16
<b>MSI clock ranges</b>	
LL_RCC_MSIRANGE_0	MSI = 65.536 KHz
LL_RCC_MSIRANGE_1	MSI = 131.072 KHz
LL_RCC_MSIRANGE_2	MSI = 262.144 KHz
LL_RCC_MSIRANGE_3	MSI = 524.288 KHz
LL_RCC_MSIRANGE_4	MSI = 1.048 MHz
LL_RCC_MSIRANGE_5	MSI = 2.097 MHz
LL_RCC_MSIRANGE_6	MSI = 4.194 MHz
<b>Oscillator Values adaptation</b>	
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
HSI48_VALUE	Value of the HSI48 oscillator in Hz
<b>Peripheral clock frequency</b>	
LL_RCC_PERIPH_FREQUENCY_NO	No clock enabled for the peripheral
LL_RCC_PERIPH_FREQUENCY_NA	Frequency cannot be provided as external clock
<b>PLL SOURCE</b>	
LL_RCC_PLLSOURCE_HSI	HSI clock selected as PLL entry clock source

`LL_RCC_PLLSOURCE_HSE` HSE clock selected as PLL entry clock source

***PLL division factor***

`LL_RCC_PLL_DIV_2` PLL clock output = PLLVCO / 2

`LL_RCC_PLL_DIV_3` PLL clock output = PLLVCO / 3

`LL_RCC_PLL_DIV_4` PLL clock output = PLLVCO / 4

***PLL Multiplicator factor***

`LL_RCC_PLL_MUL_3` PLL input clock \* 3

`LL_RCC_PLL_MUL_4` PLL input clock \* 4

`LL_RCC_PLL_MUL_6` PLL input clock \* 6

`LL_RCC_PLL_MUL_8` PLL input clock \* 8

`LL_RCC_PLL_MUL_12` PLL input clock \* 12

`LL_RCC_PLL_MUL_16` PLL input clock \* 16

`LL_RCC_PLL_MUL_24` PLL input clock \* 24

`LL_RCC_PLL_MUL_32` PLL input clock \* 32

`LL_RCC_PLL_MUL_48` PLL input clock \* 48

***Peripheral RNG get clock source***

`LL_RCC_RNG_CLKSOURCE` HSI48 RC clock source selection bit for RNG

***Peripheral RNG clock source selection***

`LL_RCC_RNG_CLKSOURCE_PLL` PLL selected as RNG clock

`LL_RCC_RNG_CLKSOURCE_HSI48` HSI48 selected as RNG clock

***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` HSE oscillator clock divided by a programmable prescaler (selection through

***RTC HSE Prescaler***

`LL_RCC_RTC_HSE_DIV_2` HSE is divided by 2 for RTC clock

`LL_RCC_RTC_HSE_DIV_4` HSE is divided by 4 for RTC clock

`LL_RCC_RTC_HSE_DIV_8` HSE is divided by 8 for RTC clock

`LL_RCC_RTC_HSE_DIV_16` HSE is divided by 16 for RTC clock

***Wakeup from Stop and CSS backup clock selection***

`LL_RCC_STOP_WAKEUPCLOCK_MSI` MSI selection after wake-up from STOP

`LL_RCC_STOP_WAKEUPCLOCK_HSI` HSI selection after wake-up from STOP

***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_MSI	MSI selection as system clock
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_MSI	MSI used as system clock
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 USART get clock source**

LL_RCC_USART1_CLKSOURCE	USART1 clock source selection bits
LL_RCC_USART2_CLKSOURCE	USART2 clock source selection bits

**Peripheral USART clock source selection**

LL_RCC_USART1_CLKSOURCE_PCLK2	PCLK2 selected as USART1 clock
LL_RCC_USART1_CLKSOURCE_SYSCLK	SYSCLK selected as USART1 clock
LL_RCC_USART1_CLKSOURCE_HSI	HSI selected as USART1 clock
LL_RCC_USART1_CLKSOURCE_LSE	LSE selected as USART1 clock
LL_RCC_USART2_CLKSOURCE_PCLK1	PCLK1 selected as USART2 clock
LL_RCC_USART2_CLKSOURCE_SYSCLK	SYSCLK selected as USART2 clock
LL_RCC_USART2_CLKSOURCE_HSI	HSI selected as USART2 clock
LL_RCC_USART2_CLKSOURCE_LSE	LSE selected as USART2 clock

**Peripheral USB get clock source**

LL_RCC_USB_CLKSOURCE	HSI48 RC clock source selection bit for USB
----------------------	---

**Peripheral USB clock source selection**

LL_RCC_USB_CLKSOURCE_PLL	PLL selected as USB clock
LL_RCC_USB_CLKSOURCE_HSI48	HSI48 selected as USB clock

**Calculate frequencies**

__LL_RCC_CALC_PLLCLK_F	Description:
REQ	

- Helper macro to calculate the PLLCLK frequency.

**Parameters:**

- \_\_INPUTFREQ\_\_: PLL Input frequency (based on MSI/HSE/HSI)
- \_\_PLLMUL\_\_: This parameter can be one of the following values:
  - LL\_RCC\_PLL\_MUL\_3
  - LL\_RCC\_PLL\_MUL\_4
  - LL\_RCC\_PLL\_MUL\_6
  - LL\_RCC\_PLL\_MUL\_8
  - LL\_RCC\_PLL\_MUL\_12
  - LL\_RCC\_PLL\_MUL\_16
  - LL\_RCC\_PLL\_MUL\_24
  - LL\_RCC\_PLL\_MUL\_32
  - LL\_RCC\_PLL\_MUL\_48
- \_\_PLLDIV\_\_: This parameter can be one of the following values:
  - LL\_RCC\_PLL\_DIV\_2
  - LL\_RCC\_PLL\_DIV\_3
  - LL\_RCC\_PLL\_DIV\_4

**Return value:**

- PLL: clock frequency (in Hz)

**Notes:**

- ex: \_\_LL\_RCC\_CALC\_PLLCLK\_FREQ  
(HSE\_VALUE, LL\_RCC\_PLL\_GetMultiplicator (),  
LL\_RCC\_PLL\_GetDivider ());

\_\_LL\_RCC\_CALC\_HCLK\_FR  
EQ

**Description:**

- Helper macro to calculate the HCLK frequency.

**Parameters:**

- \_\_SYSCLKFREQ\_\_: SYSCLK frequency (based on MSI/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_GetAHB  
Prescaler())`

`__LL_RCC_CALC_PCLK1_F  
REQ`

#### Description:

- Helper macro to calculate the PCLK1 frequency (APB1)

#### 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_GetAPB1Prescaler())`

`__LL_RCC_CALC_PCLK2_F  
REQ`

#### Description:

- Helper macro to calculate the PCLK2 frequency (APB2)

#### 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())`

`__LL_RCC_CALC_MSI_FRE  
Q`

#### Description:

- Helper macro to calculate the MSI frequency (in Hz)

#### Parameters:

- `__MSIRANGE__`: This parameter can be one of the following values:
  - `LL_RCC_MSIRANGE_0`
  - `LL_RCC_MSIRANGE_1`
  - `LL_RCC_MSIRANGE_2`
  - `LL_RCC_MSIRANGE_3`
  - `LL_RCC_MSIRANGE_4`
  - `LL_RCC_MSIRANGE_5`
  - `LL_RCC_MSIRANGE_6`

**Return value:**

- MSI: clock frequency (in Hz)

**Notes:**

- : `__MSIRANGE__` can be retrieved by `LL_RCC_MSI_GetRange` ex:  
`__LL_RCC_CALC_MSI_FREQ(LL_RCC_MSI_GetRange())`

**Common Write and read registers Macros**

<code>LL_RCC_WriteReg</code>	<b>Description:</b>
	<ul style="list-style-type: none"><li>• Write a value in RCC register.</li></ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"><li>• <code>__REG__</code>: Register to be written</li><li>• <code>__VALUE__</code>: Value to be written in the register</li></ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"><li>• None</li></ul>
<code>LL_RCC_ReadReg</code>	<b>Description:</b>
	<ul style="list-style-type: none"><li>• Read a value in RCC register.</li></ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"><li>• <code>__REG__</code>: Register to be read</li></ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"><li>• Register: value</li></ul>

## 70 LL RNG Generic Driver

### 70.1 RNG Firmware driver API description

#### 70.1.1 Detailed description of functions

##### **LL\_RNG\_Enable**

Function Name	<code>__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)</code>
Function Description	Enable Random Number Generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR RNGEN LL_RNG_Enable</li> </ul>

##### **LL\_RNG\_Disable**

Function Name	<code>__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)</code>
Function Description	Disable Random Number Generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR RNGEN LL_RNG_Disable</li> </ul>

##### **LL\_RNG\_IsEnabled**

Function Name	<code>__STATIC_INLINE uint32_t LL_RNG_IsEnabled (RNG_TypeDef * RNGx)</code>
Function Description	Check if Random Number Generator is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR RNGEN LL_RNG_IsEnabled</li> </ul>

##### **LL\_RNG\_IsActiveFlag\_DRDY**

Function Name	<code>__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (RNG_TypeDef * RNGx)</code>
Function Description	Indicate if the RNG Data ready Flag is set or not.

Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR DRDY LL_RNG_IsActiveFlag_DRDY</li> </ul>

### LL\_RNG\_IsActiveFlag\_CECS

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (RNG_TypeDef * RNGx)</code></b>
Function Description	Indicate if the Clock Error Current Status Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR CECS LL_RNG_IsActiveFlag_CECS</li> </ul>

### LL\_RNG\_IsActiveFlag\_SECS

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (RNG_TypeDef * RNGx)</code></b>
Function Description	Indicate if the Seed Error Current Status Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR SECS LL_RNG_IsActiveFlag_SECS</li> </ul>

### LL\_RNG\_IsActiveFlag\_CEIS

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (RNG_TypeDef * RNGx)</code></b>
Function Description	Indicate if the Clock Error Interrupt Status Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR CEIS LL_RNG_IsActiveFlag_CEIS</li> </ul>

### LL\_RNG\_IsActiveFlag\_SEIS

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (RNG_TypeDef * RNGx)</code></b>
Function Description	Indicate if the Seed Error Interrupt Status Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR SEIS LL_RNG_IsActiveFlag_SEIS</li> </ul>

### LL\_RNG\_ClearFlag\_CEIS

Function Name	<b><u>__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)</u></b>
Function Description	Clear Clock Error interrupt Status (CEIS) Flag.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR CEIS LL_RNG_ClearFlag_CEIS</li> </ul>

### LL\_RNG\_ClearFlag\_SEIS

Function Name	<b><u>__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)</u></b>
Function Description	Clear Seed Error interrupt Status (SEIS) Flag.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR SEIS LL_RNG_ClearFlag_SEIS</li> </ul>

### LL\_RNG\_EnableIT

Function Name	<b><u>__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)</u></b>
Function Description	Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR IE LL_RNG_EnableIT</li> </ul>

### LL\_RNG\_DisableIT

Function Name	<b><u>__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)</u></b>
Function Description	Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR IE LL_RNG_DisableIT</li> </ul>

### LL\_RNG\_IsEnabledIT

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (RNG_TypeDef * RNGx)</code></b>
Function Description	Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR IE LL_RNG_IsEnabledIT</li> </ul>

### LL\_RNG\_ReadRandData32

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (RNG_TypeDef * RNGx)</code></b>
Function Description	Return32-bit Random Number value.
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Generated:</b> 32-bit random value</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR RNDATA LL_RNG_ReadRandData32</li> </ul>

### LL\_RNG\_DeInit

Function Name	<b><code>ErrorStatus LL_RNG_DeInit (RNG_TypeDef * RNGx)</code></b>
Function Description	De-initialize RNG registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none"> <li><b>RNGx:</b> RNG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>SUCCESS: RNG registers are de-initialized</li> <li>ERROR: not applicable</li> </ul> </li> </ul>

## 70.2 RNG Firmware driver defines

### 70.2.1 RNG

#### *Get Flags Defines*

<code>LL_RNG_SR_DRDY</code>	Register contains valid random data
<code>LL_RNG_SR_CECS</code>	Clock error current status
<code>LL_RNG_SR_SECS</code>	Seed error current status

LL_RNG_SR_CEIS	Clock error interrupt status
LL_RNG_SR_SEIS	Seed error interrupt status

**IT Defines**

LL_RNG_CR_IE	RNG Interrupt enable
--------------	----------------------

**Common Write and read registers Macros**

LL_RNG_WriteReg	<b>Description:</b>
-----------------	---------------------

- Write a value in RNG register.

**Parameters:**

- \_\_INSTANCE\_\_: RNG Instance
- \_\_REG\_\_: Register to be written
- \_\_VALUE\_\_: Value to be written in the register

**Return value:**

- None

LL_RNG_ReadReg
----------------

**Description:**

- Read a value in RNG register.

**Parameters:**

- \_\_INSTANCE\_\_: RNG Instance
- \_\_REG\_\_: Register to be read

**Return value:**

- Register: value

## 71 LL RTC Generic Driver

### 71.1 RTC Firmware driver registers structures

#### 71.1.1 LL\_RTC\_InitTypeDef

##### Data Fields

- *uint32\_t HourFormat*
- *uint32\_t AsynchPrescaler*
- *uint32\_t SynchPrescaler*

##### Field Documentation

- ***uint32\_t LL\_RTC\_InitTypeDef::HourFormat***  
Specifies the RTC Hours Format. This parameter can be a value of **RTC\_LL\_EC\_HOURFORMAT**This feature can be modified afterwards using unitary function **LL\_RTC\_SetHourFormat()**.
- ***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 = 0x7FThis feature can be modified afterwards using unitary function **LL\_RTC\_SetAsynchPrescaler()**.
- ***uint32\_t LL\_RTC\_InitTypeDef::SynchPrescaler***  
Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFFFFThis feature can be modified afterwards using unitary function **LL\_RTC\_SetSynchPrescaler()**.

#### 71.1.2 LL\_RTC\_TimeTypeDef

##### Data Fields

- *uint32\_t TimeFormat*
- *uint8\_t Hours*
- *uint8\_t Minutes*
- *uint8\_t Seconds*

##### Field Documentation

- ***uint32\_t LL\_RTC\_TimeTypeDef::TimeFormat***  
Specifies the RTC AM/PM Time. This parameter can be a value of **RTC\_LL\_EC\_TIME\_FORMAT**This feature can be modified afterwards using unitary function **LL\_RTC\_TIME\_SetFormat()**.
- ***uint8\_t LL\_RTC\_TimeTypeDef::Hours***  
Specifies the RTC Time Hours. This parameter must be a number between Min\_Data = 0 and Max\_Data = 12 if the **LL\_RTC\_TIME\_FORMAT\_PM** is selected. This parameter must be a number between Min\_Data = 0 and Max\_Data = 23 if the **LL\_RTC\_TIME\_FORMAT\_AM\_OR\_24** is selected.This feature can be modified afterwards using unitary function **LL\_RTC\_TIME\_SetHour()**.

- ***uint8\_t LL\_RTC\_TimeTypeDef::Minutes***  
Specifies the RTC Time Minutes. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59This feature can be modified afterwards using unitary function **LL\_RTC\_TIME\_SetMinute()**.
- ***uint8\_t LL\_RTC\_TimeTypeDef::Seconds***  
Specifies the RTC Time Seconds. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59This feature can be modified afterwards using unitary function **LL\_RTC\_TIME\_SetSecond()**.

### 71.1.3 LL\_RTC\_DateTypeDef

#### Data Fields

- ***uint8\_t WeekDay***
- ***uint8\_t Month***
- ***uint8\_t Day***
- ***uint8\_t Year***

#### Field Documentation

- ***uint8\_t LL\_RTC\_DateTypeDef::WeekDay***  
Specifies the RTC Date WeekDay. This parameter can be a value of **RTC\_LL\_EC\_WEEKDAY**This feature can be modified afterwards using unitary function **LL\_RTC\_DATE\_SetWeekDay()**.
- ***uint8\_t LL\_RTC\_DateTypeDef::Month***  
Specifies the RTC Date Month. This parameter can be a value of **RTC\_LL\_EC\_MONTH**This feature can be modified afterwards using unitary function **LL\_RTC\_DATE\_SetMonth()**.
- ***uint8\_t LL\_RTC\_DateTypeDef::Day***  
Specifies the RTC Date Day. This parameter must be a number between Min\_Data = 1 and Max\_Data = 31This feature can be modified afterwards using unitary function **LL\_RTC\_DATE\_SetDay()**.
- ***uint8\_t LL\_RTC\_DateTypeDef::Year***  
Specifies the RTC Date Year. This parameter must be a number between Min\_Data = 0 and Max\_Data = 99This feature can be modified afterwards using unitary function **LL\_RTC\_DATE\_SetYear()**.

### 71.1.4 LL\_RTC\_AlarmTypeDef

#### Data Fields

- ***LL\_RTC\_TimeTypeDef AlarmTime***
- ***uint32\_t AlarmMask***
- ***uint32\_t AlarmDateWeekDaySel***
- ***uint8\_t AlarmDateWeekDay***

#### Field Documentation

- **`LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime`**  
Specifies the RTC Alarm Time members.
- **`uint32_t LL_RTC_AlarmTypeDef::AlarmMask`**  
Specifies the RTC Alarm Masks. This parameter can be a value of `RTC_LL_EC_ALMA_MASK` for ALARM A or `RTC_LL_EC_ALMB_MASK` for ALARM B. This feature can be modified afterwards using unitary function `LL_RTC_ALMA_SetMask()` for ALARM A or `LL_RTC_ALMB_SetMask()` for ALARM B
- **`uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel`**  
Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of `RTC_LL_EC_ALMA_WEEKDAY_SELECTION` for ALARM A or `RTC_LL_EC_ALMB_WEEKDAY_SELECTION` for ALARM B. This feature can be modified afterwards using unitary function `LL_RTC_ALMA_EnableWeekday()` or `LL_RTC_ALMA_DisableWeekday()` for ALARM A or `LL_RTC_ALMB_EnableWeekday()` or `LL_RTC_ALMB_DisableWeekday()` for ALARM B
- **`uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay`**  
Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min\_Data = 1 and Max\_Data = 31. This feature can be modified afterwards using unitary function `LL_RTC_ALMA_SetDay()` for ALARM A or `LL_RTC_ALMB_SetDay()` for ALARM B. If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of `RTC_LL_EC_WEEKDAY`. This feature can be modified afterwards using unitary function `LL_RTC_ALMA_SetWeekDay()` for ALARM A or `LL_RTC_ALMB_SetWeekDay()` for ALARM B.

## 71.2 RTC Firmware driver API description

### 71.2.1 Detailed description of functions

#### LL\_RTC\_SetHourFormat

Function Name	<code>__STATIC_INLINE void LL_RTC_SetHourFormat( RTC_TypeDef * RTCx, uint32_t HourFormat)</code>
Function Description	Set Hours format (24 hour/day or AM/PM hour format)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>HourFormat:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_RTC_HOURFORMAT_24HOUR</code></li> <li>- <code>LL_RTC_HOURFORMAT_AMPM</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. <code>LL_RTC_DisableWriteProtection</code> function should be called before.</li> <li>• It can be written in initialization mode only (<code>LL_RTC_EnableInitMode</code> function)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR FMT LL_RTC_SetHourFormat</li> </ul>

**LL\_RTC\_GetHourFormat**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_GetHourFormat(     RTC_TypeDef * RTCx)</code>
Function Description	Get Hours format (24 hour/day or AM/PM hour format)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_HOURFORMAT_24HOUR</li> <li>- LL_RTC_HOURFORMAT_AMPM</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR FMT LL_RTC_GetHourFormat</li> </ul>

**LL\_RTC\_SetAlarmOutEvent**

Function Name	<code>__STATIC_INLINE void LL_RTC_SetAlarmOutEvent(     RTC_TypeDef * RTCx, uint32_t AlarmOutput)</code>
Function Description	Select the flag to be routed to RTC_ALARM output.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>AlarmOutput:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_ALARMOUT_DISABLE</li> <li>- LL_RTC_ALARMOUT_ALMA</li> <li>- LL_RTC_ALARMOUT_ALMB</li> <li>- LL_RTC_ALARMOUT_WAKEUP</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR OSEL LL_RTC_SetAlarmOutEvent</li> </ul>

**LL\_RTC\_GetAlarmOutEvent**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent(     RTC_TypeDef * RTCx)</code>
Function Description	Get the flag to be routed to RTC_ALARM output.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_ALARMOUT_DISABLE</li> <li>- LL_RTC_ALARMOUT_ALMA</li> <li>- LL_RTC_ALARMOUT_ALMB</li> <li>- LL_RTC_ALARMOUT_WAKEUP</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR OSEL LL_RTC_GetAlarmOutEvent</li> </ul>

**LL\_RTC\_SetAlarmOutputType**

Function Name	<code>__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)</code>
Function Description	Set RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Output:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN</li> <li>– LL_RTC_ALARM_OUTPUTTYPE_PUSH_PULL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Used only when RTC_ALARM is mapped on PC13</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• OR ALARMOUTTYPE LL_RTC_SetAlarmOutputType</li> </ul>

**LL\_RTC\_GetAlarmOutputType**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (RTC_TypeDef * RTCx)</code>
Function Description	Get RTC_ALARM output type (ALARM in push-pull or open-drain output)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN</li> <li>– LL_RTC_ALARM_OUTPUTTYPE_PUSH_PULL</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• used only when RTC_ALARM is mapped on PC13</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• OR ALARMOUTTYPE LL_RTC_GetAlarmOutputType</li> </ul>

**LL\_RTC\_EnableInitMode**

Function Name	<code>__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)</code>
Function Description	Enable initialization mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR INIT LL_RTC_EnableInitMode</li> </ul>

**LL\_RTC\_DisableInitMode**

Function Name      **`__STATIC_INLINE void LL_RTC_DisableInitMode(RTC_TypeDef * RTCx)`**

Function Description      Disable initialization mode (Free running mode)

Parameters      • **RTCx:** RTC Instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
• ISR INIT LL\_RTC\_DisableInitMode

**LL\_RTC\_SetOutputPolarity**

Function Name      **`__STATIC_INLINE void LL_RTC_SetOutputPolarity(RTC_TypeDef * RTCx, uint32_t Polarity)`**

Function Description      Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is asserted)

Parameters      • **RTCx:** RTC Instance  
• **Polarity:** This parameter can be one of the following values:  
– LL\_RTC\_OUTPUTPOLARITY\_PIN\_HIGH  
– LL\_RTC\_OUTPUTPOLARITY\_PIN\_LOW

Return values      • **None:**

Notes      • Bit is write-protected. LL\_RTC\_DisableWriteProtection function should be called before.

Reference Manual to  
LL API cross  
reference:  
• CR POL LL\_RTC\_SetOutputPolarity

**LL\_RTC\_GetOutputPolarity**

Function Name      **`__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity(RTC_TypeDef * RTCx)`**

Function Description      Get Output polarity.

Parameters      • **RTCx:** RTC Instance

Return values      • **Returned:** value can be one of the following values:  
– LL\_RTC\_OUTPUTPOLARITY\_PIN\_HIGH  
– LL\_RTC\_OUTPUTPOLARITY\_PIN\_LOW

Reference Manual to  
LL API cross  
reference:  
• CR POL LL\_RTC\_GetOutputPolarity

**LL\_RTC\_EnableShadowRegBypass**

Function Name      **`__STATIC_INLINE void LL_RTC_EnableShadowRegBypass(RTC_TypeDef * RTCx)`**

Function Description      Enable Bypass the shadow registers.

Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR BYPSHAD LL_RTC_EnableShadowRegBypass</li> </ul>

### LL\_RTC\_DisableShadowRegBypass

Function Name	<code>__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)</code>
Function Description	Disable Bypass the shadow registers.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR BYPSHAD LL_RTC_DisableShadowRegBypass</li> </ul>

### LL\_RTC\_IsShadowRegBypassEnabled

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (RTC_TypeDef * RTCx)</code>
Function Description	Check if Shadow registers bypass is enabled or not.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled</li> </ul>

### LL\_RTC\_EnableRefClock

Function Name	<code>__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)</code>
Function Description	Enable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR REFCKON LL_RTC_EnableRefClock</li> </ul>

**LL\_RTC\_DisableRefClock**

Function Name	<b>_STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)</b>
Function Description	Disable RTC_REFIN reference clock detection (50 or 60 Hz)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>• It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> <li>• CR REFCKON LL_RTC_DisableRefClock</li> </ul>
Reference Manual to LL API cross reference:	

**LL\_RTC\_SetAsynchPrescaler**

Function Name	<b>_STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)</b>
Function Description	Set Asynchronous prescaler factor.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>AsynchPrescaler:</b> Value between Min_Data = 0 and Max_Data = 0x7F</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PRER_PREDIV_A LL_RTC_SetAsynchPrescaler</li> </ul>

**LL\_RTC\_SetSynchPrescaler**

Function Name	<b>_STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)</b>
Function Description	Set Synchronous prescaler factor.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>SynchPrescaler:</b> Value between Min_Data = 0 and Max_Data = 0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PRER_PREDIV_S LL_RTC_SetSynchPrescaler</li> </ul>

**LL\_RTC\_GetAsynchPrescaler**

Function Name	<b>_STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (RTC_TypeDef * RTCx)</b>
Function Description	Get Asynchronous prescaler factor.

Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data = 0 and Max_Data = 0x7F</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>PRER_PREDIV_A LL_RTC_GetAsynchPrescaler</li> </ul>

### LL\_RTC\_GetSynchPrescaler

Function Name	<b><u>STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler( RTC_TypeDef * RTCx )</u></b>
Function Description	Get Synchronous prescaler factor.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data = 0 and Max_Data = 0x7FFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>PRER_PREDIV_S LL_RTC_GetSynchPrescaler</li> </ul>

### LL\_RTC\_EnableWriteProtection

Function Name	<b><u>STATIC_INLINE void LL_RTC_EnableWriteProtection( RTC_TypeDef * RTCx )</u></b>
Function Description	Enable the write protection for RTC registers.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>WPR KEY LL_RTC_EnableWriteProtection</li> </ul>

### LL\_RTC\_DisableWriteProtection

Function Name	<b><u>STATIC_INLINE void LL_RTC_DisableWriteProtection( RTC_TypeDef * RTCx )</u></b>
Function Description	Disable the write protection for RTC registers.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>WPR KEY LL_RTC_DisableWriteProtection</li> </ul>

### LL\_RTC\_EnableOutRemap

Function Name	<b><u>STATIC_INLINE void LL_RTC_EnableOutRemap( RTC_TypeDef * RTCx )</u></b>
Function Description	Enable RTC_OUT remap.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OR OUT_RMP LL_RTC_EnableOutRemap</li> </ul>

### LL\_RTC\_DisableOutRemap

Function Name	<code>__STATIC_INLINE void LL_RTC_DisableOutRemap(     RTC_TypeDef * RTCx)</code>
Function Description	Disable RTC_OUT remap.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>OR OUT_RMP LL_RTC_DisableOutRemap</li> </ul>

### LL\_RTC\_TIME\_SetFormat

Function Name	<code>__STATIC_INLINE void LL_RTC_TIME_SetFormat(     RTC_TypeDef * RTCx, uint32_t TimeFormat)</code>
Function Description	Set time format (AM/24-hour or PM notation)
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>TimeFormat:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_RTC_TIME_FORMAT_AM_OR_24</li> <li>– LL_RTC_TIME_FORMAT_PM</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TR PM LL_RTC_TIME_SetFormat</li> </ul>

### LL\_RTC\_TIME\_GetFormat

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat(     RTC_TypeDef * RTCx)</code>
Function Description	Get time format (AM or PM notation)
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_RTC_TIME_FORMAT_AM_OR_24</li> <li>– LL_RTC_TIME_FORMAT_PM</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>Read either RTC_SSR or RTC_TR locks the values in the</li> </ul>

Reference Manual to  
LL API cross  
reference:

higher-order calendar shadow registers until RTC\_DR is read  
(LL\_RTC\_ReadReg(RTC, DR)).

- TR PM LL\_RTC\_TIME\_SetFormat

### LL\_RTC\_TIME\_SetHour

Function Name	<code>_STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)</code>
Function Description	Set Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Hours:</b> Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>• It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TR HT LL_RTC_TIME_SetHour</li> <li>• TR HU LL_RTC_TIME_SetHour</li> </ul>

### LL\_RTC\_TIME\_GetHour

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (RTC_TypeDef * RTCx)</code>
Function Description	Get Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>• Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).</li> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TR HT LL_RTC_TIME_GetHour</li> <li>• TR HU LL_RTC_TIME_GetHour</li> </ul>

### LL\_RTC\_TIME\_SetMinute

Function Name	<code>_STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)</code>
---------------	---

Function Description	Set Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Minutes:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> <li>helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TR MNT LL_RTC_TIME_SetMinute</li> <li>TR MNU LL_RTC_TIME_SetMinute</li> </ul>

### LL\_RTC\_TIME\_GetMinute

Function Name	<b><u>STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute( RTC_TypeDef * RTCx )</u></b>
Function Description	Get Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).</li> <li>helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TR MNT LL_RTC_TIME_GetMinute</li> <li>TR MNU LL_RTC_TIME_GetMinute</li> </ul>

### LL\_RTC\_TIME\_SetSecond

Function Name	<b><u>STATIC_INLINE void LL_RTC_TIME_SetSecond( RTC_TypeDef * RTCx, uint32_t Seconds )</u></b>
Function Description	Set Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Seconds:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> <li>helper macro __LL_RTC_CONVERT_BIN2BCD is available</li> </ul>

to convert Seconds from binary to BCD format

Reference Manual to  
LL API cross  
reference:

- TR ST LL\_RTC\_TIME\_SetSecond
- TR SU LL\_RTC\_TIME\_SetSecond

### **LL\_RTC\_TIME\_GetSecond**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (RTC_TypeDef * RTCx)</code>
Function Description	Get Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>• Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).</li> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TR ST LL_RTC_TIME_GetSecond</li> <li>• TR SU LL_RTC_TIME_GetSecond</li> </ul>

### **LL\_RTC\_TIME\_Config**

Function Name	<code>__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)</code>
Function Description	Set time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Format12_24:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_TIME_FORMAT_AM_OR_24</li> <li>- LL_RTC_TIME_FORMAT_PM</li> </ul> </li> <li>• <b>Hours:</b> Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> <li>• <b>Minutes:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> <li>• <b>Seconds:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>• It can be written in initialization mode only (LL_RTC_EnableInitMode function)</li> <li>• TimeFormat and Hours should follow the same format</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• TR PM LL_RTC_TIME_Config</li> <li>• TR HT LL_RTC_TIME_Config</li> </ul>

- reference:
- TR HU LL\_RTC\_TIME\_Config
  - TR MNT LL\_RTC\_TIME\_Config
  - TR MNU LL\_RTC\_TIME\_Config
  - TR ST LL\_RTC\_TIME\_Config
  - TR SU LL\_RTC\_TIME\_Config

### **LL\_RTC\_TIME\_Get**

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TIME_Get (RTC_TypeDef * RTCx)</code>
Function Description	Get time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Combination:</b> of hours, minutes and seconds (Format: 0x00HHMMSS).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>• Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).</li> <li>• helper macros <code>_LL_RTC_GET_HOUR</code>, <code>_LL_RTC_GET_MINUTE</code> and <code>_LL_RTC_GET_SECOND</code> are available to get independently each parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TR HT LL_RTC_TIME_Get</li> <li>• TR HU LL_RTC_TIME_Get</li> <li>• TR MNT LL_RTC_TIME_Get</li> <li>• TR MNU LL_RTC_TIME_Get</li> <li>• TR ST LL_RTC_TIME_Get</li> <li>• TR SU LL_RTC_TIME_Get</li> </ul>

### **LL\_RTC\_TIME\_EnableDayLightStore**

Function Name	<code>_STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)</code>
Function Description	Memorize whether the daylight saving time change has been performed.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR BCK LL_RTC_TIME_EnableDayLightStore</li> </ul>

### **LL\_RTC\_TIME\_DisableDayLightStore**

Function Name	<code>_STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)</code>
Function Description	Disable memorization whether the daylight saving time change

---

	has been performed.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR BCK LL_RTC_TIME_DisableDayLightStore</li> </ul>

### LL\_RTC\_TIME\_IsDayLightStoreEnabled

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (RTC_TypeDef * RTCx)</code>
Function Description	Check if RTC Day Light Saving stored operation has been enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR BCK LL_RTC_TIME_IsDayLightStoreEnabled</li> </ul>

### LL\_RTC\_TIME\_DecHour

Function Name	<code>__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)</code>
Function Description	Subtract 1 hour (winter time change)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR SUB1H LL_RTC_TIME_DecHour</li> </ul>

### LL\_RTC\_TIME\_IncHour

Function Name	<code>__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)</code>
Function Description	Add 1 hour (summer time change)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>

Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR ADD1H LL_RTC_TIME_IncHour</li> </ul>
---	--

**LL\_RTC\_TIME\_GetSubSecond**

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond(RTC_TypeDef * RTCx)</code>
Function Description	Get Sub second value in the synchronous prescaler counter.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Sub:</b> second value (number between 0 and 65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula:  <math display="block">\text{Seconds fraction ratio} * \text{time\_unit} = [(\text{SecondFraction} - \text{SubSeconds}) / (\text{SecondFraction} + 1)] * \text{time\_unit}</math> <p>This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S &gt;= SS.</p> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SSR SS LL_RTC_TIME_GetSubSecond</li> </ul>

**LL\_RTC\_TIME\_Synchronize**

Function Name	<code>_STATIC_INLINE void LL_RTC_TIME_Synchronize(RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)</code>
Function Description	Synchronize to a remote clock with a high degree of precision.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>ShiftSecond:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RTC_SHIFT_SECOND_DELAY</li> <li>– LL_RTC_SHIFT_SECOND_ADVANCE</li> </ul> </li> <li><b>Fraction:</b> Number of Seconds Fractions (any value from 0 to 0x7FFF)</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This operation effectively subtracts from (delays) or advance the clock of a fraction of a second.</li> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>When REFCKON is set, firmware must not write to Shift control register.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SHIFTR ADD1S LL_RTC_TIME_Synchronize</li> <li>SHIFTR SUBFS LL_RTC_TIME_Synchronize</li> </ul>

**LL\_RTC\_DATE\_SetYear**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)</code></b>
Function Description	Set Year in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Year:</b> Value between Min_Data=0x00 and Max_Data=0x99</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR YT LL_RTC_DATE_SetYear</li> <li>• DR YU LL_RTC_DATE_SetYear</li> </ul>

**LL\_RTC\_DATE\_GetYear**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Year in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x99</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR YT LL_RTC_DATE_GetYear</li> <li>• DR YU LL_RTC_DATE_GetYear</li> </ul>

**LL\_RTC\_DATE\_SetWeekDay**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)</code></b>
Function Description	Set Week day.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>WeekDay:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_WEEKDAY_MONDAY</li> <li>- LL_RTC_WEEKDAY_TUESDAY</li> <li>- LL_RTC_WEEKDAY_WEDNESDAY</li> <li>- LL_RTC_WEEKDAY_THURSDAY</li> <li>- LL_RTC_WEEKDAY_FRIDAY</li> <li>- LL_RTC_WEEKDAY_SATURDAY</li> <li>- LL_RTC_WEEKDAY_SUNDAY</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• DR WDU LL_RTC_DATE_SetWeekDay</li> </ul>

reference:

### **LL\_RTC\_DATE\_GetWeekDay**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay(RTC_TypeDef * RTCx)</code></b>
Function Description	Get Week day.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_WEEKDAY_MONDAY</li> <li>- LL_RTC_WEEKDAY_TUESDAY</li> <li>- LL_RTC_WEEKDAY_WEDNESDAY</li> <li>- LL_RTC_WEEKDAY_THURSDAY</li> <li>- LL_RTC_WEEKDAY_FRIDAY</li> <li>- LL_RTC_WEEKDAY_SATURDAY</li> <li>- LL_RTC_WEEKDAY_SUNDAY</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR WDU LL_RTC_DATE_GetWeekDay</li> </ul>

### **LL\_RTC\_DATE\_SetMonth**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_DATE_SetMonth(RTC_TypeDef * RTCx, uint32_t Month)</code></b>
Function Description	Set Month in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Month:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_MONTH_JANUARY</li> <li>- LL_RTC_MONTH_FEBRUARY</li> <li>- LL_RTC_MONTH_MARCH</li> <li>- LL_RTC_MONTH_APRIIL</li> <li>- LL_RTC_MONTH_MAY</li> <li>- LL_RTC_MONTH_JUNE</li> <li>- LL_RTC_MONTH_JULY</li> <li>- LL_RTC_MONTH_AUGUST</li> <li>- LL_RTC_MONTH_SEPTEMBER</li> <li>- LL_RTC_MONTH_OCTOBER</li> <li>- LL_RTC_MONTH_NOVEMBER</li> <li>- LL_RTC_MONTH_DECEMBER</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro <code>_LL_RTC_CONVERT_BIN2BCD</code> is available to convert Month from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR MT LL_RTC_DATE_SetMonth</li> <li>• DR MU LL_RTC_DATE_SetMonth</li> </ul>

**LL\_RTC\_DATE\_GetMonth**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (RTC_TypeDef * RTCx)</code>
Function Description	Get Month in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_MONTH_JANUARY</li> <li>- LL_RTC_MONTH_FEBRUARY</li> <li>- LL_RTC_MONTH_MARCH</li> <li>- LL_RTC_MONTH_APRI</li> <li>- LL_RTC_MONTH_MAY</li> <li>- LL_RTC_MONTH_JUNE</li> <li>- LL_RTC_MONTH_JULY</li> <li>- LL_RTC_MONTH_AUGUST</li> <li>- LL_RTC_MONTH_SEPTMBER</li> <li>- LL_RTC_MONTH_OCTOBER</li> <li>- LL_RTC_MONTH_NOVEMBER</li> <li>- LL_RTC_MONTH_DECEMBER</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR MT LL_RTC_DATE_GetMonth</li> <li>• DR MU LL_RTC_DATE_GetMonth</li> </ul>

**LL\_RTC\_DATE\_SetDay**

Function Name	<code>__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)</code>
Function Description	Set Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Day:</b> Value between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR DT LL_RTC_DATE_SetDay</li> <li>• DR DU LL_RTC_DATE_SetDay</li> </ul>

**LL\_RTC\_DATE\_GetDay**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (RTC_TypeDef * RTCx)</code>
Function Description	Get Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Notes	<ul style="list-style-type: none"> <li>if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DT LL_RTC_DATE_GetDay</li> <li>DR DU LL_RTC_DATE_GetDay</li> </ul>

### LL\_RTC\_DATE\_Config

Function Name	<pre><code>__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)</code></pre>
Function Description	Set date (WeekDay, Day, Month and Year) in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>WeekDay:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_WEEKDAY_MONDAY</li> <li>- LL_RTC_WEEKDAY_TUESDAY</li> <li>- LL_RTC_WEEKDAY_WEDNESDAY</li> <li>- LL_RTC_WEEKDAY_THURSDAY</li> <li>- LL_RTC_WEEKDAY_FRIDAY</li> <li>- LL_RTC_WEEKDAY_SATURDAY</li> <li>- LL_RTC_WEEKDAY_SUNDAY</li> </ul> </li> <li><b>Day:</b> Value between Min_Data=0x01 and Max_Data=0x31</li> <li><b>Month:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_MONTH_JANUARY</li> <li>- LL_RTC_MONTH_FEBRUARY</li> <li>- LL_RTC_MONTH_MARCH</li> <li>- LL_RTC_MONTH_APRIIL</li> <li>- LL_RTC_MONTH_MAY</li> <li>- LL_RTC_MONTH_JUNE</li> <li>- LL_RTC_MONTH_JULY</li> <li>- LL_RTC_MONTH_AUGUST</li> <li>- LL_RTC_MONTH_SEPTEMBER</li> <li>- LL_RTC_MONTH_OCTOBER</li> <li>- LL_RTC_MONTH_NOVEMBER</li> <li>- LL_RTC_MONTH_DECEMBER</li> </ul> </li> <li><b>Year:</b> Value between Min_Data=0x00 and Max_Data=0x99</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR WDU LL_RTC_DATE_Config</li> <li>DR MT LL_RTC_DATE_Config</li> <li>DR MU LL_RTC_DATE_Config</li> <li>DR DT LL_RTC_DATE_Config</li> <li>DR DU LL_RTC_DATE_Config</li> <li>DR YT LL_RTC_DATE_Config</li> <li>DR YU LL_RTC_DATE_Config</li> </ul>

**LL\_RTC\_DATE\_Get**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_DATE_Get (RTC_TypeDef * RTCx)</code>
Function Description	Get date (WeekDay, Day, Month and Year) in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Combination:</b> of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit</li> <li>• helper macros <code>__LL_RTC_GET_WEEKDAY</code>, <code>__LL_RTC_GET_YEAR</code>, <code>__LL_RTC_GET_MONTH</code>, and <code>__LL_RTC_GET_DAY</code> are available to get independently each parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DR WDU LL_RTC_DATE_Get</li> <li>• DR MT LL_RTC_DATE_Get</li> <li>• DR MU LL_RTC_DATE_Get</li> <li>• DR DT LL_RTC_DATE_Get</li> <li>• DR DU LL_RTC_DATE_Get</li> <li>• DR YT LL_RTC_DATE_Get</li> <li>• DR YU LL_RTC_DATE_Get</li> </ul>

**LL\_RTC\_ALMA\_Enable**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)</code>
Function Description	Enable Alarm A.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRAE LL_RTC_ALMA_Enable</li> </ul>

**LL\_RTC\_ALMA\_Disable**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)</code>
Function Description	Disable Alarm A.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR ALRAE LL_RTC_ALMA_Disable</li> </ul>

reference:

### **LL\_RTC\_ALMA\_SetMask**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)</code></b>
Function Description	Specify the Alarm A masks.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Mask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_ALMA_MASK_NONE</li> <li>- LL_RTC_ALMA_MASK_DATEWEEKDAY</li> <li>- LL_RTC_ALMA_MASK_HOURS</li> <li>- LL_RTC_ALMA_MASK_MINUTES</li> <li>- LL_RTC_ALMA_MASK_SECONDS</li> <li>- LL_RTC_ALMA_MASK_ALL</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR MSK4 LL_RTC_ALMA_SetMask</li> <li>• ALRMAR MSK3 LL_RTC_ALMA_SetMask</li> <li>• ALRMAR MSK2 LL_RTC_ALMA_SetMask</li> <li>• ALRMAR MSK1 LL_RTC_ALMA_SetMask</li> </ul>

### **LL\_RTC\_ALMA\_GetMask**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (RTC_TypeDef * RTCx)</code></b>
Function Description	Get the Alarm A masks.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_ALMA_MASK_NONE</li> <li>- LL_RTC_ALMA_MASK_DATEWEEKDAY</li> <li>- LL_RTC_ALMA_MASK_HOURS</li> <li>- LL_RTC_ALMA_MASK_MINUTES</li> <li>- LL_RTC_ALMA_MASK_SECONDS</li> <li>- LL_RTC_ALMA_MASK_ALL</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR MSK4 LL_RTC_ALMA_GetMask</li> <li>• ALRMAR MSK3 LL_RTC_ALMA_GetMask</li> <li>• ALRMAR MSK2 LL_RTC_ALMA_GetMask</li> <li>• ALRMAR MSK1 LL_RTC_ALMA_GetMask</li> </ul>

### **LL\_RTC\_ALMA\_EnableWeekday**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable AlarmA Week day selection (DU[3:0] represents the week day.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday</li> </ul>

### LL\_RTC\_ALMA\_DisableWeekday

Function Name	<b><code>__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday( RTC_TypeDef * RTCx )</code></b>
Function Description	Disable AlarmA Week day selection (DU[3:0] represents the date )
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday</li> </ul>

### LL\_RTC\_ALMA\_SetDay

Function Name	<b><code>__STATIC_INLINE void LL_RTC_ALMA_SetDay( RTC_TypeDef * RTCx, uint32_t Day )</code></b>
Function Description	Set ALARM A Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Day:</b> Value between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ALRMAR DT LL_RTC_ALMA_SetDay</li> <li>ALRMAR DU LL_RTC_ALMA_SetDay</li> </ul>

### LL\_RTC\_ALMA\_GetDay

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay( RTC_TypeDef * RTCx )</code></b>
Function Description	Get ALARM A Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Notes	<ul style="list-style-type: none"> <li>helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ALRMAR DT LL_RTC_ALMA_GetDay</li> <li>ALRMAR DU LL_RTC_ALMA_GetDay</li> </ul>

### LL\_RTC\_ALMA\_SetWeekDay

Function Name	<b><code>__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay</code></b>
---------------	---

**(RTC\_TypeDef \* RTCx, uint32\_t WeekDay)**

Function Description	Set ALARM A Weekday.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>WeekDay:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_WEEKDAY_MONDAY</li> <li>- LL_RTC_WEEKDAY_TUESDAY</li> <li>- LL_RTC_WEEKDAY_WEDNESDAY</li> <li>- LL_RTC_WEEKDAY_THURSDAY</li> <li>- LL_RTC_WEEKDAY_FRIDAY</li> <li>- LL_RTC_WEEKDAY_SATURDAY</li> <li>- LL_RTC_WEEKDAY_SUNDAY</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR DU LL_RTC_ALMA_SetWeekDay</li> </ul>

**LL\_RTC\_ALMA\_GetWeekDay**

Function Name	<b>_STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (RTC_TypeDef * RTCx)</b>
Function Description	Get ALARM A Weekday.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_WEEKDAY_MONDAY</li> <li>- LL_RTC_WEEKDAY_TUESDAY</li> <li>- LL_RTC_WEEKDAY_WEDNESDAY</li> <li>- LL_RTC_WEEKDAY_THURSDAY</li> <li>- LL_RTC_WEEKDAY_FRIDAY</li> <li>- LL_RTC_WEEKDAY_SATURDAY</li> <li>- LL_RTC_WEEKDAY_SUNDAY</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR DU LL_RTC_ALMA_SetWeekDay</li> </ul>

**LL\_RTC\_ALMA\_SetTimeFormat**

Function Name	<b>_STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)</b>
Function Description	Set Alarm A time format (AM/24-hour or PM notation)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>TimeFormat:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_ALMA_TIME_FORMAT_AM</li> <li>- LL_RTC_ALMA_TIME_FORMAT_PM</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• ALRMAR PM LL_RTC_ALMA_SetTimeFormat</li> </ul>

reference:

### **LL\_RTC\_ALMA\_GetTimeFormat**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Alarm A time format (AM or PM notation)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_ALMA_TIME_FORMAT_AM</li> <li>- LL_RTC_ALMA_TIME_FORMAT_PM</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR PM LL_RTC_ALMA_GetTimeFormat</li> </ul>

### **LL\_RTC\_ALMA\_SetHour**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)</code></b>
Function Description	Set ALARM A Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Hours:</b> Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR HT LL_RTC_ALMA_SetHour</li> <li>• ALRMAR HU LL_RTC_ALMA_SetHour</li> </ul>

### **LL\_RTC\_ALMA\_GetHour**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (RTC_TypeDef * RTCx)</code></b>
Function Description	Get ALARM A Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR HT LL_RTC_ALMA_GetHour</li> <li>• ALRMAR HU LL_RTC_ALMA_GetHour</li> </ul>

**LL\_RTC\_ALMA\_SetMinute**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)</code>
Function Description	Set ALARM A Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Minutes:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR MNT LL_RTC_ALMA_SetMinute</li> <li>• ALRMAR MNU LL_RTC_ALMA_SetMinute</li> </ul>

**LL\_RTC\_ALMA\_GetMinute**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (RTC_TypeDef * RTCx)</code>
Function Description	Get ALARM A Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR MNT LL_RTC_ALMA_GetMinute</li> <li>• ALRMAR MNU LL_RTC_ALMA_GetMinute</li> </ul>

**LL\_RTC\_ALMA\_SetSecond**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)</code>
Function Description	Set ALARM A Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Seconds:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR ST LL_RTC_ALMA_SetSecond</li> <li>• ALRMAR SU LL_RTC_ALMA_SetSecond</li> </ul>

**LL\_RTC\_ALMA\_GetSecond**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond</code>
---------------	---

**(RTC\_TypeDef \* RTCx)**

Function Description	Get ALARM A Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR ST LL_RTC_ALMA_GetSecond</li> <li>• ALRMAR SU LL_RTC_ALMA_GetSecond</li> </ul>

**LL\_RTC\_ALMA\_ConfigTime**

Function Name	<b>_STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)</b>
Function Description	Set Alarm A Time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Format12_24:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_ALMA_TIME_FORMAT_AM</li> <li>- LL_RTC_ALMA_TIME_FORMAT_PM</li> </ul> </li> <li>• <b>Hours:</b> Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> <li>• <b>Minutes:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> <li>• <b>Seconds:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMAR PM LL_RTC_ALMA_ConfigTime</li> <li>• ALRMAR HT LL_RTC_ALMA_ConfigTime</li> <li>• ALRMAR HU LL_RTC_ALMA_ConfigTime</li> <li>• ALRMAR MNT LL_RTC_ALMA_ConfigTime</li> <li>• ALRMAR MNU LL_RTC_ALMA_ConfigTime</li> <li>• ALRMAR ST LL_RTC_ALMA_ConfigTime</li> <li>• ALRMAR SU LL_RTC_ALMA_ConfigTime</li> </ul>

**LL\_RTC\_ALMA\_GetTime**

Function Name	<b>_STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (RTC_TypeDef * RTCx)</b>
Function Description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Combination:</b> of hours, minutes and seconds.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• ALRMAR HT LL_RTC_ALMA_GetTime</li> </ul>

- LL API cross reference:
- ALRMAR HU LL\_RTC\_ALMA\_GetTime
  - ALRMAR MNT LL\_RTC\_ALMA\_GetTime
  - ALRMAR MNU LL\_RTC\_ALMA\_GetTime
  - ALRMAR ST LL\_RTC\_ALMA\_GetTime
  - ALRMAR SU LL\_RTC\_ALMA\_GetTime

### **LL\_RTC\_ALMA\_SetSubSecondMask**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask(RTC_TypeDef * RTCx, uint32_t Mask)</code>
Function Description	Set Alarm A Mask the most-significant bits starting at this bit.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Mask:</b> Value between Min_Data=0x00 and Max_Data=0xF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask</li> </ul>

### **LL\_RTC\_ALMA\_GetSubSecondMask**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask(RTC_TypeDef * RTCx)</code>
Function Description	Get Alarm A Mask the most-significant bits starting at this bit.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask</li> </ul>

### **LL\_RTC\_ALMA\_SetSubSecond**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond(RTC_TypeDef * RTCx, uint32_t Subsecond)</code>
Function Description	Set Alarm A Sub seconds value.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Subsecond:</b> Value between Min_Data=0x00 and Max_Data=0x7FFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMASSR SS LL_RTC_ALMA_SetSubSecond</li> </ul>

### **LL\_RTC\_ALMA\_GetSubSecond**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond</code>
---------------	--

**(RTC\_TypeDef \* RTCx)**

Function Description	Get Alarm A Sub seconds value.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x7FFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMASSR SS LL_RTC_ALMA_GetSubSecond</li> </ul>

**LL\_RTC\_ALMB\_Enable**

Function Name	<b><u>__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)</u></b>
Function Description	Enable Alarm B.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRBE LL_RTC_ALMB_Enable</li> </ul>

**LL\_RTC\_ALMB\_Disable**

Function Name	<b><u>__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)</u></b>
Function Description	Disable Alarm B.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRBE LL_RTC_ALMB_Disable</li> </ul>

**LL\_RTC\_ALMB\_SetMask**

Function Name	<b><u>__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)</u></b>
Function Description	Specify the Alarm B masks.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Mask:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_ALMB_MASK_NONE</li> <li>- LL_RTC_ALMB_MASK_DATEWEEKDAY</li> <li>- LL_RTC_ALMB_MASK_HOURS</li> <li>- LL_RTC_ALMB_MASK_MINUTES</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_RTC_ALMB_MASK_SECONDS</li> <li>- LL_RTC_ALMB_MASK_ALL</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR MSK4 LL_RTC_ALMB_SetMask</li> <li>• ALRMBR MSK3 LL_RTC_ALMB_SetMask</li> <li>• ALRMBR MSK2 LL_RTC_ALMB_SetMask</li> <li>• ALRMBR MSK1 LL_RTC_ALMB_SetMask</li> </ul>

### LL\_RTC\_ALMB\_GetMask

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask( (RTC_TypeDef * RTCx)</code></b>
Function Description	Get the Alarm B masks.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be can be a combination of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_ALMB_MASK_NONE</li> <li>- LL_RTC_ALMB_MASK_DATEWEEKDAY</li> <li>- LL_RTC_ALMB_MASK_HOURS</li> <li>- LL_RTC_ALMB_MASK_MINUTES</li> <li>- LL_RTC_ALMB_MASK_SECONDS</li> <li>- LL_RTC_ALMB_MASK_ALL</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR MSK4 LL_RTC_ALMB_GetMask</li> <li>• ALRMBR MSK3 LL_RTC_ALMB_GetMask</li> <li>• ALRMBR MSK2 LL_RTC_ALMB_GetMask</li> <li>• ALRMBR MSK1 LL_RTC_ALMB_GetMask</li> </ul>

### LL\_RTC\_ALMB\_EnableWeekday

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMB_EnableWeekday( (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable AlarmB Week day selection (DU[3:0] represents the week day).
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday</li> </ul>

### LL\_RTC\_ALMB\_DisableWeekday

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMB_DisableWeekday( (RTC_TypeDef * RTCx)</code></b>
Function Description	Disable AlarmB Week day selection (DU[3:0] represents the date )
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- Reference Manual to  
LL API cross  
reference:
- ALRMBR WDSEL LL\_RTC\_ALMB\_DisableWeekday

### LL\_RTC\_ALMB\_SetDay

Function Name	<code>_STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)</code>
Function Description	Set ALARM B Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Day:</b> Value between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR DT LL_RTC_ALMB_SetDay</li> <li>• ALRMBR DU LL_RTC_ALMB_SetDay</li> </ul>

### LL\_RTC\_ALMB\_GetDay

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (RTC_TypeDef * RTCx)</code>
Function Description	Get ALARM B Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR DT LL_RTC_ALMB_GetDay</li> <li>• ALRMBR DU LL_RTC_ALMB_GetDay</li> </ul>

### LL\_RTC\_ALMB\_SetWeekDay

Function Name	<code>_STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)</code>
Function Description	Set ALARM B Weekday.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>WeekDay:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_WEEKDAY_MONDAY</li> <li>- LL_RTC_WEEKDAY_TUESDAY</li> <li>- LL_RTC_WEEKDAY_WEDNESDAY</li> <li>- LL_RTC_WEEKDAY_THURSDAY</li> <li>- LL_RTC_WEEKDAY_FRIDAY</li> <li>- LL_RTC_WEEKDAY_SATURDAY</li> <li>- LL_RTC_WEEKDAY_SUNDAY</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to  
LL API cross  
reference:

- ALRMBR DU LL\_RTC\_ALMB\_SetWeekDay

### **LL\_RTC\_ALMB\_GetWeekDay**

Function Name      **`_STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay  
(RTC_TypeDef * RTCx)`**

Function Description      Get ALARM B Weekday.

Parameters      • **RTCx:** RTC Instance

Return values      • **Returned:** value can be one of the following values:

- LL\_RTC\_WEEKDAY\_MONDAY
- LL\_RTC\_WEEKDAY\_TUESDAY
- LL\_RTC\_WEEKDAY\_WEDNESDAY
- LL\_RTC\_WEEKDAY\_THURSDAY
- LL\_RTC\_WEEKDAY\_FRIDAY
- LL\_RTC\_WEEKDAY\_SATURDAY
- LL\_RTC\_WEEKDAY\_SUNDAY

Reference Manual to  
LL API cross  
reference:

- ALRMBR DU LL\_RTC\_ALMB\_GetWeekDay

### **LL\_RTC\_ALMB\_SetTimeFormat**

Function Name      **`_STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat  
(RTC_TypeDef * RTCx, uint32_t TimeFormat)`**

Function Description      Set ALARM B time format (AM/24-hour or PM notation)

Parameters      • **RTCx:** RTC Instance

                  • **TimeFormat:** This parameter can be one of the following values:

- LL\_RTC\_ALMB\_TIME\_FORMAT\_AM
- LL\_RTC\_ALMB\_TIME\_FORMAT\_PM

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:

- ALRMBR PM LL\_RTC\_ALMB\_SetTimeFormat

### **LL\_RTC\_ALMB\_GetTimeFormat**

Function Name      **`_STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat  
(RTC_TypeDef * RTCx)`**

Function Description      Get ALARM B time format (AM or PM notation)

Parameters      • **RTCx:** RTC Instance

Return values      • **Returned:** value can be one of the following values:

- LL\_RTC\_ALMB\_TIME\_FORMAT\_AM
- LL\_RTC\_ALMB\_TIME\_FORMAT\_PM

Reference Manual to  
LL API cross

- ALRMBR PM LL\_RTC\_ALMB\_GetTimeFormat

reference:

### **LL\_RTC\_ALMB\_SetHour**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)</code></b>
Function Description	Set ALARM B Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Hours:</b> Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR HT LL_RTC_ALMB_SetHour</li> <li>• ALRMBR HU LL_RTC_ALMB_SetHour</li> </ul>

### **LL\_RTC\_ALMB\_GetHour**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (RTC_TypeDef * RTCx)</code></b>
Function Description	Get ALARM B Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR HT LL_RTC_ALMB_GetHour</li> <li>• ALRMBR HU LL_RTC_ALMB_GetHour</li> </ul>

### **LL\_RTC\_ALMB\_SetMinute**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)</code></b>
Function Description	Set ALARM B Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Minutes:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR MNT LL_RTC_ALMB_SetMinute</li> <li>• ALRMBR MNU LL_RTC_ALMB_SetMinute</li> </ul>

**LL\_RTC\_ALMB\_GetMinute**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (RTC_TypeDef * RTCx)</code>
Function Description	Get ALARM B Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR MNT LL_RTC_ALMB_GetMinute</li> <li>• ALRMBR MNU LL_RTC_ALMB_GetMinute</li> </ul>

**LL\_RTC\_ALMB\_SetSecond**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)</code>
Function Description	Set ALARM B Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Seconds:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR ST LL_RTC_ALMB_SetSecond</li> <li>• ALRMBR SU LL_RTC_ALMB_SetSecond</li> </ul>

**LL\_RTC\_ALMB\_GetSecond**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (RTC_TypeDef * RTCx)</code>
Function Description	Get ALARM B Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBR ST LL_RTC_ALMB_GetSecond</li> <li>• ALRMBR SU LL_RTC_ALMB_GetSecond</li> </ul>

**LL\_RTC\_ALMB\_ConfigTime**

Function Name	<code>__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)</code>
---------------	---

Function Description	Set Alarm B Time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Format12_24:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_ALMB_TIME_FORMAT_AM</li> <li>- LL_RTC_ALMB_TIME_FORMAT_PM</li> </ul> </li> <li><b>Hours:</b> Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> <li><b>Minutes:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> <li><b>Seconds:</b> Value between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ALRMBR PM LL_RTC_ALMB_ConfigTime</li> <li>ALRMBR HT LL_RTC_ALMB_ConfigTime</li> <li>ALRMBR HU LL_RTC_ALMB_ConfigTime</li> <li>ALRMBR MNT LL_RTC_ALMB_ConfigTime</li> <li>ALRMBR MNU LL_RTC_ALMB_ConfigTime</li> <li>ALRMBR ST LL_RTC_ALMB_ConfigTime</li> <li>ALRMBR SU LL_RTC_ALMB_ConfigTime</li> </ul>

### LL\_RTC\_ALMB\_GetTime

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (RTC_TypeDef * RTCx)</code>
Function Description	Get Alarm B Time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Combination:</b> of hours, minutes and seconds.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>helper macros <code>_LL_RTC_GET_HOUR</code>, <code>_LL_RTC_GET_MINUTE</code> and <code>_LL_RTC_GET_SECOND</code> are available to get independently each parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>ALRMBR HT LL_RTC_ALMB_GetTime</li> <li>ALRMBR HU LL_RTC_ALMB_GetTime</li> <li>ALRMBR MNT LL_RTC_ALMB_GetTime</li> <li>ALRMBR MNU LL_RTC_ALMB_GetTime</li> <li>ALRMBR ST LL_RTC_ALMB_GetTime</li> <li>ALRMBR SU LL_RTC_ALMB_GetTime</li> </ul>

### LL\_RTC\_ALMB\_SetSubSecondMask

Function Name	<code>_STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)</code>
Function Description	Set Alarm B Mask the most-significant bits starting at this bit.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Mask:</b> Value between Min_Data=0x00 and Max_Data=0xF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This register can be written only when ALRBE is reset in</li> </ul>

RTC\_CR register, or in initialization mode.

- Reference Manual to LL API cross reference:
- ALRMBSSR MASKSS LL\_RTC\_ALMB\_SetSubSecondMask

### **LL\_RTC\_ALMB\_SetSubSecondMask**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Alarm B Mask the most-significant bits starting at this bit.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask</li> </ul>

### **LL\_RTC\_ALMB\_SetSubSecond**

Function Name	<b><code>__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)</code></b>
Function Description	Set Alarm B Sub seconds value.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Subsecond:</b> Value between Min_Data=0x00 and Max_Data=0x7FFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBSSR SS LL_RTC_ALMB_SetSubSecond</li> </ul>

### **LL\_RTC\_ALMB\_GetSubSecond**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Alarm B Sub seconds value.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x7FFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ALRMBSSR SS LL_RTC_ALMB_SetSubSecond</li> </ul>

### **LL\_RTC\_TS\_Enable**

Function Name	<b><code>__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable Timestamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR TSE LL_RTC_TS_Enable</li> </ul>

### LL\_RTC\_TS\_Disable

Function Name	<code>_STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)</code>
Function Description	Disable Timestamp.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR TSE LL_RTC_TS_Disable</li> </ul>

### LL\_RTC\_TS\_SetActiveEdge

Function Name	<code>_STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)</code>
Function Description	Set Time-stamp event active edge.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Edge:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_TIMESTAMP_EDGE_RISING</li> <li>- LL_RTC_TIMESTAMP_EDGE_FALLING</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR TSEDGE LL_RTC_TS_SetActiveEdge</li> </ul>

### LL\_RTC\_TS\_GetActiveEdge

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (RTC_TypeDef * RTCx)</code>
Function Description	Get Time-stamp event active edge.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_TIMESTAMP_EDGE_RISING</li> </ul> </li> </ul>

---

	– LL_RTC_TIMESTAMP_EDGE_FALLING
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR TSEDGE LL_RTC_TS_GetActiveEdge</li> </ul>

### LL\_RTC\_TS\_GetTimeFormat

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Timestamp AM/PM notation (AM or 24-hour format)
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_RTC_TS_TIME_FORMAT_AM</li> <li>– LL_RTC_TS_TIME_FORMAT_PM</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TSTR PM LL_RTC_TS_GetTimeFormat</li> </ul>

### LL\_RTC\_TS\_GetHour

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TS_GetHour (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Timestamp Hours in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23</li> </ul>
Notes	<ul style="list-style-type: none"> <li>helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TSTR HT LL_RTC_TS_GetHour</li> <li>TSTR HU LL_RTC_TS_GetHour</li> </ul>

### LL\_RTC\_TS\_GetMinute

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Timestamp Minutes in BCD format.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>TSTR MNT LL_RTC_TS_GetMinute</li> <li>TSTR MNU LL_RTC_TS_GetMinute</li> </ul>

reference:

### **LL\_RTC\_TS\_GetSecond**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Timestamp Seconds in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x59</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Seconds from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TSTR ST LL_RTC_TS_GetSecond</li> <li>• TSTR SU LL_RTC_TS_GetSecond</li> </ul>

### **LL\_RTC\_TS\_GetTime**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TS_GetTime (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Timestamp time (hour, minute and second) in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Combination:</b> of hours, minutes and seconds.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macros <code>__LL_RTC_GET_HOUR</code>, <code>__LL_RTC_GET_MINUTE</code> and <code>__LL_RTC_GET_SECOND</code> are available to get independently each parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TSTR HT LL_RTC_TS_GetTime</li> <li>• TSTR HU LL_RTC_TS_GetTime</li> <li>• TSTR MNT LL_RTC_TS_GetTime</li> <li>• TSTR MNU LL_RTC_TS_GetTime</li> <li>• TSTR ST LL_RTC_TS_GetTime</li> <li>• TSTR SU LL_RTC_TS_GetTime</li> </ul>

### **LL\_RTC\_TS\_GetWeekDay**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Timestamp Week day.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_RTC_WEEKDAY_MONDAY</code></li> <li>- <code>LL_RTC_WEEKDAY_TUESDAY</code></li> <li>- <code>LL_RTC_WEEKDAY_WEDNESDAY</code></li> <li>- <code>LL_RTC_WEEKDAY_THURSDAY</code></li> <li>- <code>LL_RTC_WEEKDAY_FRIDAY</code></li> <li>- <code>LL_RTC_WEEKDAY_SATURDAY</code></li> <li>- <code>LL_RTC_WEEKDAY_SUNDAY</code></li> </ul> </li> </ul>

- Reference Manual to  
LL API cross  
reference:
- TSDR WDU LL\_RTC\_TS\_GetWeekDay

### LL\_RTC\_TS\_GetMonth

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (RTC_TypeDef * RTCx)</code>
Function Description	Get Timestamp Month in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_MONTH_JANUARY</li> <li>- LL_RTC_MONTH_FEBRUARY</li> <li>- LL_RTC_MONTH_MARCH</li> <li>- LL_RTC_MONTH_APRI</li> <li>- LL_RTC_MONTH_MAY</li> <li>- LL_RTC_MONTH_JUNE</li> <li>- LL_RTC_MONTH_JULY</li> <li>- LL_RTC_MONTH_AUGUST</li> <li>- LL_RTC_MONTH_SEPTMBER</li> <li>- LL_RTC_MONTH_OCTOBER</li> <li>- LL_RTC_MONTH_NOVEMBER</li> <li>- LL_RTC_MONTH_DECEMBER</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Month from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TSDR MT LL_RTC_TS_GetMonth</li> <li>• TSDR MU LL_RTC_TS_GetMonth</li> </ul>

### LL\_RTC\_TS\_GetDay

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TS_GetDay (RTC_TypeDef * RTCx)</code>
Function Description	Get Timestamp Day in BCD format.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x01 and Max_Data=0x31</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• helper macro <code>__LL_RTC_CONVERT_BCD2BIN</code> is available to convert Day from BCD to Binary format</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TSDR DT LL_RTC_TS_GetDay</li> <li>• TSDR DU LL_RTC_TS_GetDay</li> </ul>

### LL\_RTC\_TS\_GetDate

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_TS_GetDate (RTC_TypeDef * RTCx)</code>
Function Description	Get Timestamp date (WeekDay, Day and Month) in BCD format.

Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Combination:</b> of Weekday, Day and Month</li> </ul>
Notes	<ul style="list-style-type: none"> <li>helper macros <code>__LL_RTC_GET_WEEKDAY</code>, <code>__LL_RTC_GET_MONTH</code>, and <code>__LL_RTC_GET_DAY</code> are available to get independently each parameter.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TSDR WDU LL_RTC_TS_GetDate</li> <li>TSDR MT LL_RTC_TS_GetDate</li> <li>TSDR MU LL_RTC_TS_GetDate</li> <li>TSDR DT LL_RTC_TS_GetDate</li> <li>TSDR DU LL_RTC_TS_GetDate</li> </ul>

### LL\_RTC\_TS\_GetSubSecond

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond( RTC_TypeDef * RTCx )</code>
Function Description	Get time-stamp sub second value.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TSSSR SS LL_RTC_TS_GetSubSecond</li> </ul>

### LL\_RTC\_TS\_EnableOnTamper

Function Name	<code>__STATIC_INLINE void LL_RTC_TS_EnableOnTamper( RTC_TypeDef * RTCx )</code>
Function Description	Activate timestamp on tamper detection event.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TAMPCR TAMPTS LL_RTC_TS_EnableOnTamper</li> </ul>

### LL\_RTC\_TS\_DisableOnTamper

Function Name	<code>__STATIC_INLINE void LL_RTC_TS_DisableOnTamper( RTC_TypeDef * RTCx )</code>
Function Description	Disable timestamp on tamper detection event.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>TAMPCR TAMPTS LL_RTC_TS_DisableOnTamper</li> </ul>

**LL\_RTC\_TAMPER\_Enable**

Function Name	<code>_STATIC_INLINE void LL_RTC_TAMPER_Enable (RTC_TypeDef * RTCx, uint32_t Tamper)</code>
Function Description	Enable RTC_TAMPx input detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Tamper:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RTC_TAMPER_1 (*)</li> <li>- LL_RTC_TAMPER_2</li> <li>- LL_RTC_TAMPER_3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1E LL_RTC_TAMPER_Enable</li> <li>• TAMPCR TAMP2E LL_RTC_TAMPER_Enable</li> <li>• TAMPCR TAMP3E LL_RTC_TAMPER_Enable</li> </ul>

**LL\_RTC\_TAMPER\_Disable**

Function Name	<code>_STATIC_INLINE void LL_RTC_TAMPER_Disable (RTC_TypeDef * RTCx, uint32_t Tamper)</code>
Function Description	Clear RTC_TAMPx input detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Tamper:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RTC_TAMPER_1 (*)</li> <li>- LL_RTC_TAMPER_2</li> <li>- LL_RTC_TAMPER_3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1E LL_RTC_TAMPER_Disable</li> <li>• TAMPCR TAMP2E LL_RTC_TAMPER_Disable</li> <li>• TAMPCR TAMP3E LL_RTC_TAMPER_Disable</li> </ul>

**LL\_RTC\_TAMPER\_EnableMask**

Function Name	<code>_STATIC_INLINE void LL_RTC_TAMPER_EnableMask (RTC_TypeDef * RTCx, uint32_t Mask)</code>
Function Description	Enable Tamper mask flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Mask:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RTC_TAMPER_MASK_TAMPER1 (*)</li> <li>- LL_RTC_TAMPER_MASK_TAMPER2</li> <li>- LL_RTC_TAMPER_MASK_TAMPER3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Associated Tamper IT must not be enabled when tamper mask is set.</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1MF LL_RTC_TAMPER_EnableMask</li> </ul>

- LL API cross reference:  
  - TAMPCR TAMP2MF LL\_RTC\_TAMPER\_EnableMask
  - TAMPCR TAMP3MF LL\_RTC\_TAMPER\_EnableMask

### **LL\_RTC\_TAMPER\_DisableMask**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_DisableMask (RTC_TypeDef * RTCx, uint32_t Mask)</code></b>
Function Description	Disable Tamper mask flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Mask:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RTC_TAMPER_MASK_TAMPER1 (*)</li> <li>- LL_RTC_TAMPER_MASK_TAMPER2</li> <li>- LL_RTC_TAMPER_MASK_TAMPER3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1MF LL_RTC_TAMPER_DisableMask</li> <li>• TAMPCR TAMP2MF LL_RTC_TAMPER_DisableMask</li> <li>• TAMPCR TAMP3MF LL_RTC_TAMPER_DisableMask</li> </ul>

### **LL\_RTC\_TAMPER\_EnableEraseBKP**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)</code></b>
Function Description	Enable backup register erase after Tamper event detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Tamper:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RTC_TAMPER_NOERASE_TAMPER1 (*)</li> <li>- LL_RTC_TAMPER_NOERASE_TAMPER2</li> <li>- LL_RTC_TAMPER_NOERASE_TAMPER3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1NOERASE</li> <li>• LL_RTC_TAMPER_EnableEraseBKP</li> <li>• TAMPCR TAMP2NOERASE</li> <li>• LL_RTC_TAMPER_EnableEraseBKP</li> <li>• TAMPCR TAMP3NOERASE</li> <li>• LL_RTC_TAMPER_EnableEraseBKP</li> </ul>

### **LL\_RTC\_TAMPER\_DisableEraseBKP**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)</code></b>
Function Description	Disable backup register erase after Tamper event detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Tamper:</b> This parameter can be a combination of the following values: (*) value not defined in all devices. <ul style="list-style-type: none"> <li>- LL_RTC_TAMPER_NOERASE_TAMPER1 (*)</li> <li>- LL_RTC_TAMPER_NOERASE_TAMPER2</li> </ul> </li> </ul>

– LL\_RTC\_TAMPER\_NOERASE\_TAMPER3 (\*)

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP</li> <li>• TAMPCR TAMP2NOERASE LL_RTC_TAMPER_DisableEraseBKP</li> <li>• TAMPCR TAMP3NOERASE LL_RTC_TAMPER_DisableEraseBKP</li> </ul>

### LL\_RTC\_TAMPER\_DisablePullUp

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (RTC_TypeDef * RTCx)</code></b>
Function Description	Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp</li> </ul>

### LL\_RTC\_TAMPER\_EnablePullUp

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable RTC_TAMPx pull-up disable ( Precharge RTC_TAMPx pins before sampling)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp</li> </ul>

### LL\_RTC\_TAMPER\_SetPrecharge

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (RTC_TypeDef * RTCx, uint32_t Duration)</code></b>
Function Description	Set RTC_TAMPx precharge duration.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Duration:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RTC_TAMPER_DURATION_1RTCCLK</li> <li>– LL_RTC_TAMPER_DURATION_2RTCCLK</li> <li>– LL_RTC_TAMPER_DURATION_4RTCCLK</li> <li>– LL_RTC_TAMPER_DURATION_8RTCCLK</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• TAMPCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge</li> </ul>

reference:

### **LL\_RTC\_TAMPER\_GetPrecharge**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge( (RTC_TypeDef * RTCx))</code>
Function Description	Get RTC_TAMPx precharge duration.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_RTC_TAMPER_DURATION_1RTCCLK</li> <li>– LL_RTC_TAMPER_DURATION_2RTCCLK</li> <li>– LL_RTC_TAMPER_DURATION_4RTCCLK</li> <li>– LL_RTC_TAMPER_DURATION_8RTCCLK</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge</li> </ul>

### **LL\_RTC\_TAMPER\_SetFilterCount**

Function Name	<code>__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount( (RTC_TypeDef * RTCx, uint32_t FilterCount)</code>
Function Description	Set RTC_TAMPx filter count.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>FilterCount:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_RTC_TAMPER_FILTER_DISABLE</li> <li>– LL_RTC_TAMPER_FILTER_2SAMPLE</li> <li>– LL_RTC_TAMPER_FILTER_4SAMPLE</li> <li>– LL_RTC_TAMPER_FILTER_8SAMPLE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPFLT LL_RTC_TAMPER_SetFilterCount</li> </ul>

### **LL\_RTC\_TAMPER\_GetFilterCount**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount( (RTC_TypeDef * RTCx))</code>
Function Description	Get RTC_TAMPx filter count.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_RTC_TAMPER_FILTER_DISABLE</li> <li>– LL_RTC_TAMPER_FILTER_2SAMPLE</li> <li>– LL_RTC_TAMPER_FILTER_4SAMPLE</li> <li>– LL_RTC_TAMPER_FILTER_8SAMPLE</li> </ul> </li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• TAMPCR TAMPFLT LL_RTC_TAMPER_GetFilterCount</li> </ul>

reference:

### **LL\_RTC\_TAMPER\_SetSamplingFreq**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (RTC_TypeDef * RTCx, uint32_t SamplingFreq)</code></b>
Function Description	Set Tamper sampling frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>SamplingFreq:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_32768</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_16384</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_8192</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_4096</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_2048</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_1024</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_512</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_256</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq</li> </ul>

### **LL\_RTC\_TAMPER\_GetSamplingFreq**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Tamper sampling frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_32768</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_16384</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_8192</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_4096</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_2048</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_1024</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_512</code></li> <li>- <code>LL_RTC_TAMPER_SAMPLFREQDIV_256</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq</li> </ul>

### **LL\_RTC\_TAMPER\_EnableActiveLevel**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)</code></b>
Function Description	Enable Active level for Tamper input.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Tamper:</b> This parameter can be a combination of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>– LL_RTC_TAMPER_ACTIVELEVEL_TAMP1 (*)</li> <li>– LL_RTC_TAMPER_ACTIVELEVEL_TAMP2</li> <li>– LL_RTC_TAMPER_ACTIVELEVEL_TAMP3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel</li> <li>• TAMPCR TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel</li> <li>• TAMPCR TAMP3TRG LL_RTC_TAMPER_EnableActiveLevel</li> </ul>

### LL\_RTC\_TAMPER\_DisableActiveLevel

Function Name	<code>__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel(RTC_TypeDef * RTCx, uint32_t Tamper)</code>
Function Description	Disable Active level for Tamper input.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Tamper:</b> This parameter can be a combination of the following values: (*) value not defined in all devices.           <ul style="list-style-type: none"> <li>– LL_RTC_TAMPER_ACTIVELEVEL_TAMP1 (*)</li> <li>– LL_RTC_TAMPER_ACTIVELEVEL_TAMP2</li> <li>– LL_RTC_TAMPER_ACTIVELEVEL_TAMP3 (*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel</li> <li>• TAMPCR TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel</li> <li>• TAMPCR TAMP3TRG LL_RTC_TAMPER_DisableActiveLevel</li> </ul>

### LL\_RTC\_WAKEUP\_Enable

Function Name	<code>__STATIC_INLINE void LL_RTC_WAKEUP_Enable(RTC_TypeDef * RTCx)</code>
Function Description	Enable Wakeup timer.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WUTE LL_RTC_WAKEUP_Enable</li> </ul>

### LL\_RTC\_WAKEUP\_Disable

Function Name	<code>__STATIC_INLINE void LL_RTC_WAKEUP_Disable(RTC_TypeDef * RTCx)</code>
---------------	---

---

Function Description	Disable Wakeup timer.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR WUTE LL_RTC_WAKEUP_Disable</li> </ul>

### LL\_RTC\_WAKEUP\_IsEnabled

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled( (RTC_TypeDef * RTCx)</code>
Function Description	Check if Wakeup timer is enabled or not.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR WUTE LL_RTC_WAKEUP_IsEnabled</li> </ul>

### LL\_RTC\_WAKEUP\_SetClock

Function Name	<code>_STATIC_INLINE void LL_RTC_WAKEUP_SetClock( (RTC_TypeDef * RTCx, uint32_t WakeupClock)</code>
Function Description	Select Wakeup clock.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>WakeupClock:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_WAKEUPCLOCK_DIV_16</li> <li>- LL_RTC_WAKEUPCLOCK_DIV_8</li> <li>- LL_RTC_WAKEUPCLOCK_DIV_4</li> <li>- LL_RTC_WAKEUPCLOCK_DIV_2</li> <li>- LL_RTC_WAKEUPCLOCK_CKSPRE</li> <li>- LL_RTC_WAKEUPCLOCK_CKSPRE_WUT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR WUCKSEL LL_RTC_WAKEUP_SetClock</li> </ul>

### LL\_RTC\_WAKEUP\_GetClock

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock( (RTC_TypeDef * RTCx)</code>
---------------	---

Function Description	Get Wakeup clock.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_RTC_WAKEUPCLOCK_DIV_16</li> <li>- LL_RTC_WAKEUPCLOCK_DIV_8</li> <li>- LL_RTC_WAKEUPCLOCK_DIV_4</li> <li>- LL_RTC_WAKEUPCLOCK_DIV_2</li> <li>- LL_RTC_WAKEUPCLOCK_CKSPRE</li> <li>- LL_RTC_WAKEUPCLOCK_CKSPRE_WUT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR WUCKSEL LL_RTC_WAKEUP_GetClock</li> </ul>

### LL\_RTC\_WAKEUP\_SetAutoReload

Function Name	<code>__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload(   RTC_TypeDef * RTCx, uint32_t Value)</code>
Function Description	Set Wakeup auto-reload value.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>Value:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Bit can be written only when WUTWF is set to 1 in RTC_ISR</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>WUTR WUT LL_RTC_WAKEUP_SetAutoReload</li> </ul>

### LL\_RTC\_WAKEUP\_GetAutoReload

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload(   RTC_TypeDef * RTCx)</code>
Function Description	Get Wakeup auto-reload value.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Value:</b> between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>

Reference Manual to  
LL API cross  
reference:

- WUTR WUT LL\_RTC\_WAKEUP\_GetAutoReload

### LL\_RTC\_BAK\_SetRegister

Function Name	<code>__STATIC_INLINE void LL_RTC_BAK_SetRegister(   RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t   Data)</code>
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>BackupRegister:</b> This parameter can be one of the following values:</li> </ul>

	<ul style="list-style-type: none"> <li>- LL_RTC_BKP_DR0</li> <li>- LL_RTC_BKP_DR1</li> <li>- LL_RTC_BKP_DR2</li> <li>- LL_RTC_BKP_DR3</li> <li>- LL_RTC_BKP_DR4</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>Data:</b> Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• BKPxR BKP LL_RTC_BAK_SetRegister</li> </ul>

### LL\_RTC\_BAK\_GetRegister

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_BAK_GetRegister(RTC_TypeDef * RTCx, uint32_t BackupRegister)</code>
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>BackupRegister:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_BKP_DR0</li> <li>- LL_RTC_BKP_DR1</li> <li>- LL_RTC_BKP_DR2</li> <li>- LL_RTC_BKP_DR3</li> <li>- LL_RTC_BKP_DR4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xFFFFFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• BKPxR BKP LL_RTC_BAK_GetRegister</li> </ul>

### LL\_RTC\_CAL\_SetOutputFreq

Function Name	<code>__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq(RTC_TypeDef * RTCx, uint32_t Frequency)</code>
Function Description	Set Calibration output frequency (1 Hz or 512 Hz)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Frequency:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_CALIB_OUTPUT_NONE</li> <li>- LL_RTC_CALIB_OUTPUT_1HZ</li> <li>- LL_RTC_CALIB_OUTPUT_512HZ</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR COE LL_RTC_CAL_SetOutputFreq</li> <li>• CR COSEL LL_RTC_CAL_SetOutputFreq</li> </ul>

**LL\_RTC\_CAL\_GetOutputFreq**

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (RTC_TypeDef * RTCx)</code>
Function Description	Get Calibration output frequency (1 Hz or 512 Hz)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_RTC_CALIB_OUTPUT_NONE</li> <li>- LL_RTC_CALIB_OUTPUT_1HZ</li> <li>- LL_RTC_CALIB_OUTPUT_512HZ</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR COE LL_RTC_CAL_GetOutputFreq</li> <li>• CR COSEL LL_RTC_CAL_GetOutputFreq</li> </ul>

**LL\_RTC\_CAL\_SetPulse**

Function Name	<code>_STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)</code>
Function Description	Insert or not One RTCCLK pulse every 2 <sup>exp11</sup> pulses (frequency increased by 488.5 ppm)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Pulse:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_RTC_CALIB_INSERTPULSE_NONE</li> <li>- LL_RTC_CALIB_INSERTPULSE_SET</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>• Bit can be written only when RECALPF is set to 0 in RTC_ISR</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALR CALP LL_RTC_CAL_SetPulse</li> </ul>

**LL\_RTC\_CAL\_IsPulseInserted**

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (RTC_TypeDef * RTCx)</code>
Function Description	Check if one RTCCLK has been inserted or not every 2 <sup>exp11</sup> pulses (frequency increased by 488.5 ppm)
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALR CALP LL_RTC_CAL_IsPulseInserted</li> </ul>

**LL\_RTC\_CAL\_SetPeriod**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)</code></b>
Function Description	Set the calibration cycle period.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>Period:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_CALIB_PERIOD_32SEC</li> <li>- LL_RTC_CALIB_PERIOD_16SEC</li> <li>- LL_RTC_CALIB_PERIOD_8SEC</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>• Bit can be written only when RECALPF is set to 0 in RTC_ISR</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALR CALW8 LL_RTC_CAL_SetPeriod</li> <li>• CALR CALW16 LL_RTC_CAL_SetPeriod</li> </ul>

**LL\_RTC\_CAL\_GetPeriod**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (RTC_TypeDef * RTCx)</code></b>
Function Description	Get the calibration cycle period.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_CALIB_PERIOD_32SEC</li> <li>- LL_RTC_CALIB_PERIOD_16SEC</li> <li>- LL_RTC_CALIB_PERIOD_8SEC</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALR CALW8 LL_RTC_CAL_GetPeriod</li> <li>• CALR CALW16 LL_RTC_CAL_GetPeriod</li> </ul>

**LL\_RTC\_CAL\_SetMinus**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)</code></b>
Function Description	Set Calibration minus.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>CalibMinus:</b> Value between Min_Data=0x00 and Max_Data=0x1FF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> <li>• Bit can be written only when RECALPF is set to 0 in RTC_ISR</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CALM CALM LL_RTC_CAL_SetMinus</li> </ul>

LL API cross  
reference:

### **LL\_RTC\_CAL\_GetMinus**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus( (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Calibration minus.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data= 0x1FF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CALR CALM LL_RTC_CAL_GetMinus</li> </ul>

### **LL\_RTC\_IsActiveFlag\_RECALP**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP( (RTC_TypeDef * RTCx)</code></b>
Function Description	Get Recalibration pending Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR RECALPF LL_RTC_IsActiveFlag_RECALP</li> </ul>

### **LL\_RTC\_IsActiveFlag\_TAMP3**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3( (RTC_TypeDef * RTCx)</code></b>
Function Description	Get RTC_TAMP3 detection flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TAMP3F LL_RTC_IsActiveFlag_TAMP3</li> </ul>

### **LL\_RTC\_IsActiveFlag\_TAMP2**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2( (RTC_TypeDef * RTCx)</code></b>
Function Description	Get RTC_TAMP2 detection flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• ISR TAMP2F LL_RTC_IsActiveFlag_TAMP2</li> </ul>

reference:

### **LL\_RTC\_IsActiveFlag\_TAMP1**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1(     RTC_TypeDef * RTCx)</code>
Function Description	Get RTC_TAMP1 detection flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TAMP1F LL_RTC_IsActiveFlag_TAMP1</li> </ul>

### **LL\_RTC\_IsActiveFlag\_TSOV**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV(     RTC_TypeDef * RTCx)</code>
Function Description	Get Time-stamp overflow flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TSOVF LL_RTC_IsActiveFlag_TSOV</li> </ul>

### **LL\_RTC\_IsActiveFlag\_TS**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS(     RTC_TypeDef * RTCx)</code>
Function Description	Get Time-stamp flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TSF LL_RTC_IsActiveFlag_TS</li> </ul>

### **LL\_RTC\_IsActiveFlag\_WUT**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT(     RTC_TypeDef * RTCx)</code>
Function Description	Get Wakeup timer flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR WUTF LL_RTC_IsActiveFlag_WUT</li> </ul>

**LL\_RTC\_IsActiveFlag\_ALRB**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (RTC_TypeDef * RTCx)</code>
Function Description	Get Alarm B flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR ALRBF LL_RTC_IsActiveFlag_ALRB</li></ul>

**LL\_RTC\_IsActiveFlag\_ALRA**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (RTC_TypeDef * RTCx)</code>
Function Description	Get Alarm A flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR ALRAF LL_RTC_IsActiveFlag_ALRA</li></ul>

**LL\_RTC\_ClearFlag\_TAMP3**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (RTC_TypeDef * RTCx)</code>
Function Description	Clear RTC_TAMP3 detection flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TAMP3F LL_RTC_ClearFlag_TAMP3</li></ul>

**LL\_RTC\_ClearFlag\_TAMP2**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)</code>
Function Description	Clear RTC_TAMP2 detection flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR TAMP2F LL_RTC_ClearFlag_TAMP2</li></ul>

**LL\_RTC\_ClearFlag\_TAMP1**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (RTC_TypeDef * RTCx)</code>
Function Description	Clear RTC_TAMP1 detection flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TAMP1F LL_RTC_ClearFlag_TAMP1</li> </ul>

**LL\_RTC\_ClearFlag\_TSOV**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)</code>
Function Description	Clear Time-stamp overflow flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TSOVF LL_RTC_ClearFlag_TSOV</li> </ul>

**LL\_RTC\_ClearFlag\_TS**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)</code>
Function Description	Clear Time-stamp flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TSF LL_RTC_ClearFlag_TS</li> </ul>

**LL\_RTC\_ClearFlag\_WUT**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)</code>
Function Description	Clear Wakeup timer flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR WUTF LL_RTC_ClearFlag_WUT</li> </ul>

**LL\_RTC\_ClearFlag\_ALRB**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)</code>
Function Description	Clear Alarm B flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR ALRBF LL_RTC_ClearFlag_ALRB</li></ul>

**LL\_RTC\_ClearFlag\_ALRA**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)</code>
Function Description	Clear Alarm A flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR ALRAF LL_RTC_ClearFlag_ALRA</li></ul>

**LL\_RTC\_IsActiveFlag\_INIT**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (RTC_TypeDef * RTCx)</code>
Function Description	Get Initialization flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR INITF LL_RTC_IsActiveFlag_INIT</li></ul>

**LL\_RTC\_IsActiveFlag\_RS**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)</code>
Function Description	Get Registers synchronization flag.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• ISR RSF LL_RTC_IsActiveFlag_RS</li></ul>

**LL\_RTC\_ClearFlag\_RS**

Function Name	<code>__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)</code>
Function Description	Clear Registers synchronization flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR RSF LL_RTC_ClearFlag_RS</li> </ul>

**LL\_RTC\_IsActiveFlag\_INITS**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (RTC_TypeDef * RTCx)</code>
Function Description	Get Initialization status flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR INITS LL_RTC_IsActiveFlag_INITS</li> </ul>

**LL\_RTC\_IsActiveFlag\_SHP**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (RTC_TypeDef * RTCx)</code>
Function Description	Get Shift operation pending flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR SHPF LL_RTC_IsActiveFlag_SHP</li> </ul>

**LL\_RTC\_IsActiveFlag\_WUTW**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)</code>
Function Description	Get Wakeup timer write flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR WUTWF LL_RTC_IsActiveFlag_WUTW</li> </ul>

**LL\_RTC\_IsActiveFlag\_ALRBW**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBW(RTC_TypeDef * RTCx)</code>
Function Description	Get Alarm B write flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ALRBWF LL_RTC_IsActiveFlag_ALRBW</li> </ul>

**LL\_RTC\_IsActiveFlag\_ALRAW**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW(RTC_TypeDef * RTCx)</code>
Function Description	Get Alarm A write flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ALRAWF LL_RTC_IsActiveFlag_ALRAW</li> </ul>

**LL\_RTC\_EnableIT\_TS**

Function Name	<code>__STATIC_INLINE void LL_RTC_EnableIT_TS(RTC_TypeDef * RTCx)</code>
Function Description	Enable Time-stamp interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR TSIE LL_RTC_EnableIT_TS</li> </ul>

**LL\_RTC\_DisableIT\_TS**

Function Name	<code>__STATIC_INLINE void LL_RTC_DisableIT_TS(RTC_TypeDef * RTCx)</code>
Function Description	Disable Time-stamp interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• CR TSIE LL_RTC_DisableIT_TS</li> </ul>

LL API cross  
reference:

### **LL\_RTC\_EnableIT\_WUT**

Function Name	<code>__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)</code>
Function Description	Enable Wakeup timer interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WUTIE LL_RTC_EnableIT_WUT</li> </ul>

### **LL\_RTC\_DisableIT\_WUT**

Function Name	<code>__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)</code>
Function Description	Disable Wakeup timer interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WUTIE LL_RTC_DisableIT_WUT</li> </ul>

### **LL\_RTC\_EnableIT\_ALRB**

Function Name	<code>__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)</code>
Function Description	Enable Alarm B interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRBIE LL_RTC_EnableIT_ALRB</li> </ul>

### **LL\_RTC\_DisableIT\_ALRB**

Function Name	<code>__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)</code>
---------------	--

Function Description	Disable Alarm B interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRBIE LL_RTC_DisableIT_ALRB</li> </ul>

### LL\_RTC\_EnableIT\_ALRA

Function Name	<b><code>_STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable Alarm A interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRAIE LL_RTC_EnableIT_ALRA</li> </ul>

### LL\_RTC\_DisableIT\_ALRA

Function Name	<b><code>_STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)</code></b>
Function Description	Disable Alarm A interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRAIE LL_RTC_DisableIT_ALRA</li> </ul>

### LL\_RTC\_EnableIT\_TAMP3

Function Name	<b><code>_STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable Tamper 3 interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• TAMPCR TAMP3IE LL_RTC_EnableIT_TAMP3</li> </ul>

reference:

### **LL\_RTC\_DisableIT\_TAMP3**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (RTC_TypeDef * RTCx)</code></b>
Function Description	Disable Tamper 3 interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP3IE LL_RTC_DisableIT_TAMP3</li> </ul>

### **LL\_RTC\_EnableIT\_TAMP2**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable Tamper 2 interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP2IE LL_RTC_EnableIT_TAMP2</li> </ul>

### **LL\_RTC\_DisableIT\_TAMP2**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (RTC_TypeDef * RTCx)</code></b>
Function Description	Disable Tamper 2 interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP2IE LL_RTC_DisableIT_TAMP2</li> </ul>

### **LL\_RTC\_EnableIT\_TAMP1**

Function Name	<b><code>_STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (RTC_TypeDef * RTCx)</code></b>
Function Description	Enable Tamper 1 interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1IE LL_RTC_EnableIT_TAMP1</li> </ul>

**LL\_RTC\_DisableIT\_TAMP1**

Function Name	<code>_STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (RTC_TypeDef * RTCx)</code>
Function Description	Disable Tamper 1 interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• TAMPCR TAMP1IE LL_RTC_DisableIT_TAMP1</li></ul>

**LL\_RTC\_EnableIT\_TAMP**

Function Name	<code>_STATIC_INLINE void LL_RTC_EnableIT_TAMP (RTC_TypeDef * RTCx)</code>
Function Description	Enable all Tamper Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• TAMPCR TAMPIE LL_RTC_EnableIT_TAMP</li></ul>

**LL\_RTC\_DisableIT\_TAMP**

Function Name	<code>_STATIC_INLINE void LL_RTC_DisableIT_TAMP (RTC_TypeDef * RTCx)</code>
Function Description	Disable all Tamper Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• TAMPCR TAMPIE LL_RTC_DisableIT_TAMP</li></ul>

**LL\_RTC\_IsEnabledIT\_TS**

Function Name	<code>_STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (RTC_TypeDef * RTCx)</code>
Function Description	Check if Time-stamp interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"><li>• <b>RTCx:</b> RTC Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR TSIE LL_RTC_IsEnabledIT_TS</li></ul>

**LL\_RTC\_IsEnabledIT\_WUT**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (RTC_TypeDef * RTCx)</code>
Function Description	Check if Wakeup timer interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WUTIE LL_RTC_IsEnabledIT_WUT</li> </ul>

**LL\_RTC\_IsEnabledIT\_ALRB**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (RTC_TypeDef * RTCx)</code>
Function Description	Check if Alarm B interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRBIE LL_RTC_IsEnabledIT_ALRB</li> </ul>

**LL\_RTC\_IsEnabledIT\_ALRA**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (RTC_TypeDef * RTCx)</code>
Function Description	Check if Alarm A interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR ALRAIE LL_RTC_IsEnabledIT_ALRA</li> </ul>

**LL\_RTC\_IsEnabledIT\_TAMP3**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (RTC_TypeDef * RTCx)</code>
Function Description	Check if Tamper 3 interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP3IE LL_RTC_IsEnabledIT_TAMP3</li> </ul>

**LL\_RTC\_IsEnabledIT\_TAMP2**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2(RTC_TypeDef * RTCx)</code>
Function Description	Check if Tamper 2 interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP2IE LL_RTC_IsEnabledIT_TAMP2</li> </ul>

**LL\_RTC\_IsEnabledIT\_TAMP1**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1(RTC_TypeDef * RTCx)</code>
Function Description	Check if Tamper 1 interrupt is enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMP1IE LL_RTC_IsEnabledIT_TAMP1</li> </ul>

**LL\_RTC\_IsEnabledIT\_TAMP**

Function Name	<code>__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP(RTC_TypeDef * RTCx)</code>
Function Description	Check if all the TAMPER interrupts are enabled or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TAMPCR TAMPIE LL_RTC_IsEnabledIT_TAMP</li> </ul>

**LL\_RTC\_DelInit**

Function Name	<code>ErrorStatus LL_RTC_DelInit (RTC_TypeDef * RTCx)</code>
Function Description	De-Initializes the RTC registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: RTC registers are de-initialized</li> <li>– ERROR: RTC registers are not de-initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function doesn't reset the RTC Clock source and RTC Backup Data registers.</li> </ul>

**LL\_RTC\_Init**

Function Name	<b>ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)</b>
Function Description	Initializes the RTC registers according to the specified parameters in RTC_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>RTC_InitStruct:</b> pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: RTC registers are initialized</li> <li>– ERROR: RTC registers are not initialized</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The RTC Prescaler register is write protected and can be written in initialization mode only.</li> </ul>

**LL\_RTC\_StructInit**

Function Name	<b>void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)</b>
Function Description	Set each LL_RTC_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTC_InitStruct:</b> pointer to a LL_RTC_InitTypeDef structure which will be initialized.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_RTC\_TIME\_Init**

Function Name	<b>ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)</b>
Function Description	Set the RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>RTC_Format:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_RTC_FORMAT_BIN</li> <li>– LL_RTC_FORMAT_BCD</li> </ul> </li> <li>• <b>RTC_TimeStruct:</b> pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>– SUCCESS: RTC Time register is configured</li> <li>– ERROR: RTC Time register is not configured</li> </ul> </li> </ul>

**LL\_RTC\_TIME\_StructInit**

Function Name	<b>void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)</b>
Function Description	Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters	<ul style="list-style-type: none"> <li><b>RTC_TimeStruct:</b> pointer to a LL_RTC_TimeTypeDef structure which will be initialized.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### LL\_RTC\_DATE\_Init

Function Name	<b>ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)</b>
Function Description	Set the RTC current date.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>RTC_Format:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_FORMAT_BIN</li> <li>- LL_RTC_FORMAT_BCD</li> </ul> </li> <li><b>RTC_DateStruct:</b> pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: RTC Day register is configured</li> <li>- ERROR: RTC Day register is not configured</li> </ul> </li> </ul>

### LL\_RTC\_DATE\_StructInit

Function Name	<b>void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)</b>
Function Description	Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)
Parameters	<ul style="list-style-type: none"> <li><b>RTC_DateStruct:</b> pointer to a LL_RTC_DateTypeDef structure which will be initialized.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

### LL\_RTC\_ALMA\_Init

Function Name	<b>ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)</b>
Function Description	Set the RTC Alarm A.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> <li><b>RTC_Format:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_FORMAT_BIN</li> <li>- LL_RTC_FORMAT_BCD</li> </ul> </li> <li><b>RTC_AlarmStruct:</b> pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: ALARMA registers are configured</li> <li>- ERROR: ALARMA registers are not configured</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>The Alarm register can only be written when the</li> </ul>

corresponding Alarm is disabled (Use LL\_RTC\_ALMA\_Disable function).

### LL\_RTC\_ALMB\_Init

Function Name	<b>ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)</b>
Function Description	Set the RTC Alarm B.
Parameters	<ul style="list-style-type: none"> <li>• <b>RTCx:</b> RTC Instance</li> <li>• <b>RTC_Format:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_RTC_FORMAT_BIN</li> <li>- LL_RTC_FORMAT_BCD</li> </ul> </li> <li>• <b>RTC_AlarmStruct:</b> pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: ALARMB registers are configured</li> <li>- ERROR: ALARMB registers are not configured</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).</li> </ul>

### LL\_RTC\_ALMA\_StructInit

Function Name	<b>void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)</b>
Function Description	Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).
Parameters	<ul style="list-style-type: none"> <li>• <b>RTC_AlarmStruct:</b> pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### LL\_RTC\_ALMB\_StructInit

Function Name	<b>void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)</b>
Function Description	Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).
Parameters	<ul style="list-style-type: none"> <li>• <b>RTC_AlarmStruct:</b> pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### LL\_RTC\_EnterInitMode

Function Name	<b>ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)</b>
---------------	--

Function Description	Enters the RTC Initialization mode.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: RTC is in Init mode</li> <li>ERROR: RTC is not in Init mode</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>The RTC Initialization mode is write protected, use the <code>LL_RTC_DisableWriteProtection</code> before calling this function.</li> </ul>

**LL\_RTC\_ExitInitMode**

Function Name	<b>ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)</b>
Function Description	Exit the RTC Initialization mode.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: RTC exited from in Init mode</li> <li>ERROR: Not applicable</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.</li> <li>The RTC Initialization mode is write protected, use the <code>LL_RTC_DisableWriteProtection</code> before calling this function.</li> </ul>

**LL\_RTC\_WaitForSynchro**

Function Name	<b>ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)</b>
Function Description	Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"> <li><b>RTCx:</b> RTC Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: RTC registers are synchronised</li> <li>ERROR: RTC registers are not synchronised</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>The RTC Resynchronization mode is write protected, use the <code>LL_RTC_DisableWriteProtection</code> before calling this function.</li> <li>To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.</li> </ul>

**71.3 RTC Firmware driver defines****71.3.1 RTC****ALARM OUTPUT**

`LL_RTC_ALARMOUT_DISABLE` Output disabled

LL_RTC_ALARMOUT_ALMA	Alarm A output enabled
LL_RTC_ALARMOUT_ALMB	Alarm B output enabled
LL_RTC_ALARMOUT_WAKEUP	Wakeup output enabled
<b>ALARM OUTPUT TYPE</b>	
LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN	RTC_ALARM, when mapped on PC13, is open-drain output
LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL	RTC_ALARM, when mapped on PC13, is push-pull output
<b>ALARMA MASK</b>	
LL_RTC_ALMA_MASK_NONE	No masks applied on Alarm A
LL_RTC_ALMA_MASK_DATEWEEKDAY	Date/day do not care in Alarm A comparison
LL_RTC_ALMA_MASK_HOURS	Hours do not care in Alarm A comparison
LL_RTC_ALMA_MASK_MINUTES	Minutes do not care in Alarm A comparison
LL_RTC_ALMA_MASK_SECONDS	Seconds do not care in Alarm A comparison
LL_RTC_ALMA_MASK_ALL	Masks all
<b>ALARMA TIME FORMAT</b>	
LL_RTC_ALMA_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_ALMA_TIME_FORMAT_PM	PM
<b>RTC Alarm A Date WeekDay</b>	
LL_RTC_ALMA_DATEWEEKDAYSEL_DATE	Alarm A Date is selected
LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY	Alarm A WeekDay is selected
<b>ALARMB MASK</b>	
LL_RTC_ALMB_MASK_NONE	No masks applied on Alarm B
LL_RTC_ALMB_MASK_DATEWEEKDAY	Date/day do not care in Alarm B comparison
LL_RTC_ALMB_MASK_HOURS	Hours do not care in Alarm B comparison
LL_RTC_ALMB_MASK_MINUTES	Minutes do not care in Alarm B comparison
LL_RTC_ALMB_MASK_SECONDS	Seconds do not care in Alarm B comparison
LL_RTC_ALMB_MASK_ALL	Masks all
<b>ALARMB TIME FORMAT</b>	
LL_RTC_ALMB_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_ALMB_TIME_FORMAT_PM	PM
<b>RTC Alarm B Date WeekDay</b>	
LL_RTC_ALMB_DATEWEEKDAYSEL_DATE	Alarm B Date is selected
LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY	Alarm B WeekDay is selected
<b>BACKUP</b>	
LL_RTC_BKP_DR0	
LL_RTC_BKP_DR1	

LL\_RTC\_BKP\_DR2

LL\_RTC\_BKP\_DR3

LL\_RTC\_BKP\_DR4

#### ***Calibration pulse insertion***

LL\_RTC\_CALIB\_INSERTPULSE\_NONE No RTCCLK pulses are added

LL\_RTC\_CALIB\_INSERTPULSE\_SET One RTCCLK pulse is effectively inserted every  $2^{exp11}$  pulses (frequency increased by 488.5 ppm)

#### ***Calibration output***

LL\_RTC\_CALIB\_OUTPUT\_NONE Calibration output disabled

LL\_RTC\_CALIB\_OUTPUT\_1HZ Calibration output is 512 Hz

LL\_RTC\_CALIB\_OUTPUT\_512HZ Calibration output is 1 Hz

#### ***Calibration period***

LL\_RTC\_CALIB\_PERIOD\_32SEC Use a 32-second calibration cycle period

LL\_RTC\_CALIB\_PERIOD\_16SEC Use a 16-second calibration cycle period

LL\_RTC\_CALIB\_PERIOD\_8SEC Use a 8-second calibration cycle period

#### ***FORMAT***

LL\_RTC\_FORMAT\_BIN Binary data format

LL\_RTC\_FORMAT\_BCD BCD data format

#### ***Get Flags Defines***

LL\_RTC\_ISR\_RECALPF

LL\_RTC\_ISR\_TAMP3F

LL\_RTC\_ISR\_TAMP2F

LL\_RTC\_ISR\_TAMP1F

LL\_RTC\_ISR\_TSOF

LL\_RTC\_ISR\_TSFI

LL\_RTC\_ISR\_WUTF

LL\_RTC\_ISR\_ALRBF

LL\_RTC\_ISR\_ALRAF

LL\_RTC\_ISR\_INITF

LL\_RTC\_ISR\_RSF

LL\_RTC\_ISR\_INITS

LL\_RTC\_ISR\_SHPF

LL\_RTC\_ISR\_WUTWF

LL\_RTC\_ISR\_ALRBWF

LL\_RTC\_ISR\_ALRAWF

#### ***HOUR FORMAT***

LL\_RTC\_HOURFORMAT\_24HOUR 24 hour/day format  
LL\_RTC\_HOURFORMAT\_AMPM AM/PM hour format

**IT Defines**

LL\_RTC\_CR\_TSIE  
LL\_RTC\_CR\_WUTIE  
LL\_RTC\_CR\_ALRBIE  
LL\_RTC\_CR\_ALRAIE  
LL\_RTC\_TAMPCR\_TAMP3IE  
LL\_RTC\_TAMPCR\_TAMP2IE  
LL\_RTC\_TAMPCR\_TAMP1IE  
LL\_RTC\_TAMPCR\_TAMPIE

**MONTH**

LL\_RTC\_MONTH\_JANUARY January  
LL\_RTC\_MONTH\_FEBRUARY February  
LL\_RTC\_MONTH\_MARCH March  
LL\_RTC\_MONTH\_APRIIL April  
LL\_RTC\_MONTH\_MAY May  
LL\_RTC\_MONTH\_JUNE June  
LL\_RTC\_MONTH\_JULY July  
LL\_RTC\_MONTH\_AUGUST August  
LL\_RTC\_MONTH\_SEPTEMBER September  
LL\_RTC\_MONTH\_OCTOBER October  
LL\_RTC\_MONTH\_NOVEMBER November  
LL\_RTC\_MONTH\_DECEMBER December

**OUTPUT POLARITY PIN**

LL\_RTC\_OUTPUTPOLARITY\_PIN\_HIGH Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)  
LL\_RTC\_OUTPUTPOLARITY\_PIN\_LOW Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

**SHIFT SECOND**

LL\_RTC\_SHIFT\_SECOND\_DELAY  
LL\_RTC\_SHIFT\_SECOND\_ADVANCE

**TAMPER**

LL\_RTC\_TAMPER\_1 RTC\_TAMP1 input detection  
LL\_RTC\_TAMPER\_2 RTC\_TAMP2 input detection  
LL\_RTC\_TAMPER\_3 RTC\_TAMP3 input detection

**TAMPER ACTIVE LEVEL**

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1	RTC_TAMP1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event
LL_RTC_TAMPER_ACTIVELEVEL_TAMP2	RTC_TAMP2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event
LL_RTC_TAMPER_ACTIVELEVEL_TAMP3	RTC_TAMP3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

**TAMPER DURATION**

LL_RTC_TAMPER_DURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
LL_RTC_TAMPER_DURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
LL_RTC_TAMPER_DURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
LL_RTC_TAMPER_DURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

**TAMPER FILTER**

LL_RTC_TAMPER_FILTER_DISABLE	Tamper filter is disabled
LL_RTC_TAMPER_FILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
LL_RTC_TAMPER_FILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
LL_RTC_TAMPER_FILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active level.

**TAMPER MASK**

LL_RTC_TAMPER_MASK_TAMPER1	Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware. The backup registers are not erased
LL_RTC_TAMPER_MASK_TAMPER2	Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.
LL_RTC_TAMPER_MASK_TAMPER3	Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

**TAMPER NO ERASE**

LL_RTC_TAMPER_NOERASE_TAMPER1	Tamper 1 event does not erase the backup registers.
LL_RTC_TAMPER_NOERASE_TAMPER2	Tamper 2 event does not erase the backup registers.
LL_RTC_TAMPER_NOERASE_TAMPER3	Tamper 3 event does not erase the backup

registers.

#### **TAMPER SAMPLING FREQUENCY DIVIDER**

LL_RTC_TAMPER_SAMPLFREQDIV_32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
LL_RTC_TAMPER_SAMPLFREQDIV_16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
LL_RTC_TAMPER_SAMPLFREQDIV_8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
LL_RTC_TAMPER_SAMPLFREQDIV_4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
LL_RTC_TAMPER_SAMPLFREQDIV_2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
LL_RTC_TAMPER_SAMPLFREQDIV_1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
LL_RTC_TAMPER_SAMPLFREQDIV_512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
LL_RTC_TAMPER_SAMPLFREQDIV_256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

#### **TIMESTAMP EDGE**

LL_RTC_TIMESTAMP_EDGE_RISING	RTC_TS input rising edge generates a timestamp event
LL_RTC_TIMESTAMP_EDGE_FALLING	RTC_TS input falling edge generates a timestamp even

#### **TIME FORMAT**

LL_RTC_TIME_FORMAT_AM_OR_24	AM or 24-hour format
LL_RTC_TIME_FORMAT_PM	PM

#### **TIMESTAMP TIME FORMAT**

LL_RTC_TS_TIME_FORMAT_AM	AM or 24-hour format
LL_RTC_TS_TIME_FORMAT_PM	PM

#### **WAKEUP CLOCK DIV**

LL_RTC_WAKEUPCLOCK_DIV_16	RTC/16 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_8	RTC/8 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_4	RTC/4 clock is selected
LL_RTC_WAKEUPCLOCK_DIV_2	RTC/2 clock is selected
LL_RTC_WAKEUPCLOCK_CKSPRE	ck_spre (usually 1 Hz) clock is selected
LL_RTC_WAKEUPCLOCK_CKSPRE_WUT	ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

#### **WEEK DAY**

LL_RTC_WEEKDAY_MONDAY	Monday
LL_RTC_WEEKDAY_TUESDAY	Tuesday

---

LL_RTC_WEEKDAY_WEDNESDAY	Wednesday
LL_RTC_WEEKDAY_THURSDAY	Thrusday
LL_RTC_WEEKDAY_FRIDAY	Friday
LL_RTC_WEEKDAY_SATURDAY	Saturday
LL_RTC_WEEKDAY_SUNDAY	Sunday

**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

**Date helper Macros****\_LL\_RTC\_GET\_WEEKDAY Description:**

- Helper macro to retrieve weekday.

**Parameters:**

- \_\_RTC\_DATE\_\_: Date returned by

**Return value:**

- Returned: value can be one of the following values:
  - LL\_RTC\_WEEKDAY\_MONDAY
  - LL\_RTC\_WEEKDAY\_TUESDAY
  - LL\_RTC\_WEEKDAY\_WEDNESDAY
  - LL\_RTC\_WEEKDAY\_THURSDAY
  - LL\_RTC\_WEEKDAY\_FRIDAY
  - LL\_RTC\_WEEKDAY\_SATURDAY
  - LL\_RTC\_WEEKDAY\_SUNDAY

**\_LL\_RTC\_GET\_YEAR Description:**

- Helper macro to retrieve Year in BCD format.

**Parameters:**

- \_\_RTC\_DATE\_\_: Value returned by

**Return value:**

- Year: in BCD format (0x00 . . . 0x99)

**\_LL\_RTC\_GET\_MONTH****Description:**

- Helper macro to retrieve Month in BCD format.

**Parameters:**

- \_RTC\_DATE\_: Value returned by

**Return value:**

- Returned: value can be one of the following values:
  - LL\_RTC\_MONTH\_JANUARY
  - LL\_RTC\_MONTH\_FEBRUARY
  - LL\_RTC\_MONTH\_MARCH
  - LL\_RTC\_MONTH\_APRI
  - LL\_RTC\_MONTH\_MAY
  - LL\_RTC\_MONTH\_JUNE
  - LL\_RTC\_MONTH\_JULY
  - LL\_RTC\_MONTH\_AUGUST
  - LL\_RTC\_MONTH\_SEPTEMBER
  - LL\_RTC\_MONTH\_OCTOBER
  - LL\_RTC\_MONTH\_NOVEMBER
  - LL\_RTC\_MONTH\_DECEMBER

**\_LL\_RTC\_GET\_DAY****Description:**

- Helper macro to retrieve Day in BCD format.

**Parameters:**

- \_RTC\_DATE\_: Value returned by

**Return value:**

- Day: in BCD format (0x01 . . . 0x31)

***Time helper Macros*****\_LL\_RTC\_GET\_HOUR****Description:**

- Helper macro to retrieve hour in BCD format.

**Parameters:**

- \_RTC\_TIME\_: RTC time returned by

**Return value:**

- Hours: in BCD format (0x01 . . . 0x12 or between Min\_Data=0x00 and Max\_Data=0x23)

**\_LL\_RTC\_GET\_MINUTE****Description:**

- Helper macro to retrieve minute in BCD format.

**Parameters:**

- \_RTC\_TIME\_: RTC time returned by

**Return value:**

- Minutes: in BCD format (0x00 . . . 0x59)

**\_LL\_RTC\_GET\_SECOND****Description:**

- Helper macro to retrieve second in BCD format.

**Parameters:**

- \_\_RTC\_TIME\_\_: RTC time returned by

**Return value:**

- Seconds: in format (0x00...0x59)

**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

## 72 LL SPI Generic Driver

### 72.1 SPI Firmware driver registers structures

#### 72.1.1 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()`.

## 72.2 SPI Firmware driver API description

### 72.2.1 Detailed description of functions

#### LL\_SPI\_Enable

Function Name	<code>__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)</code>
Function Description	Enable SPI peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 SPE LL_SPI_Enable</li> </ul>

#### LL\_SPI\_Disable

Function Name	<code>__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)</code>
Function Description	Disable SPI peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When disabling the SPI, follow the procedure described in the Reference Manual.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 SPE LL_SPI_Disable</li> </ul>

#### LL\_SPI\_IsEnabled

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)</code>
Function Description	Check if SPI peripheral is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 SPE LL_SPI_IsEnabled</li> </ul>

### LL\_SPI\_SetMode

Function Name	<code>__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)</code>
Function Description	Set SPI operation mode to Master or Slave.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>Mode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_SPI_MODE_MASTER</li> <li>LL_SPI_MODE_SLAVE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bit should not be changed when communication is ongoing.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 MSTR LL_SPI_SetMode</li> <li>CR1 SSI LL_SPI_SetMode</li> </ul>

### LL\_SPI\_GetMode

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetMode (SPI_TypeDef * SPIx)</code>
Function Description	Get SPI operation mode (Master or Slave)
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>LL_SPI_MODE_MASTER</li> <li>LL_SPI_MODE_SLAVE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 MSTR LL_SPI_GetMode</li> <li>CR1 SSI LL_SPI_GetMode</li> </ul>

### LL\_SPI\_SetStandard

Function Name	<code>__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)</code>
Function Description	Set serial protocol used.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>Standard:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_SPI_PROTOCOL_MOTOROLA</li> <li>LL_SPI_PROTOCOL_TI</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bit should be written only when SPI is disabled (SPE = 0)</li> </ul>

for correct operation.

Reference Manual to  
LL API cross  
reference:

- CR2 FRF LL\_SPI\_SetStandard

### **LL\_SPI\_GetStandard**

Function Name      **\_STATIC\_INLINE uint32\_t LL\_SPI\_GetStandard (SPI\_TypeDef \* SPIx)**

Function Description      Get serial protocol used.

Parameters      • **SPIx:** SPI Instance

Return values      • **Returned:** value can be one of the following values:  
   – LL\_SPI\_PROTOCOL\_MOTOROLA  
   – LL\_SPI\_PROTOCOL\_TI

Reference Manual to  
LL API cross  
reference:

- CR2 FRF LL\_SPI\_GetStandard

### **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  
reference:

- CR1 CPHA LL\_SPI\_SetClockPhase

### **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

- CR1 CPHA LL\_SPI\_GetClockPhase

reference:

### **LL\_SPI\_SetClockPolarity**

Function Name	<b><code>__STATIC_INLINE void LL_SPI_SetClockPolarity(SPI_TypeDef * SPIx, uint32_t ClockPolarity)</code></b>
Function Description	Set clock polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>ClockPolarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_SPI_POLARITY_LOW</code></li> <li>- <code>LL_SPI_POLARITY_HIGH</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CPOL LL_SPI_SetClockPolarity</li> </ul>

### **LL\_SPI\_GetClockPolarity**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity(SPI_TypeDef * SPIx)</code></b>
Function Description	Get clock polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_SPI_POLARITY_LOW</code></li> <li>- <code>LL_SPI_POLARITY_HIGH</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CPOL LL_SPI_GetClockPolarity</li> </ul>

### **LL\_SPI\_SetBaudRatePrescaler**

Function Name	<b><code>__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler(SPI_TypeDef * SPIx, uint32_t BaudRate)</code></b>
Function Description	Set baud rate prescaler.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>BaudRate:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV2</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV4</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV8</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV16</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV32</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV64</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV128</code></li> <li>- <code>LL_SPI_BAUDRATEPRESCALER_DIV256</code></li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 BR LL_SPI_SetBaudRatePrescaler</li> </ul>

### LL\_SPI\_GetBaudRatePrescaler

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler(                   SPI_TypeDef * SPIx)</code>
Function Description	Get baud rate prescaler.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_SPI_BAUDRATEPRESCALER_DIV2</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV4</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV8</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV16</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV32</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV64</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV128</li> <li>- LL_SPI_BAUDRATEPRESCALER_DIV256</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 BR LL_SPI_SetBaudRatePrescaler</li> </ul>

### LL\_SPI\_SetTransferBitOrder

Function Name	<code>__STATIC_INLINE void LL_SPI_SetTransferBitOrder(                   SPI_TypeDef * SPIx, uint32_t BitOrder)</code>
Function Description	Set transfer bit order.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>BitOrder:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_SPI_LSB_FIRST</li> <li>- LL_SPI_MSB_FIRST</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 LSBFIRST LL_SPI_SetTransferBitOrder</li> </ul>

### LL\_SPI\_GetTransferBitOrder

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder(                   SPI_TypeDef * SPIx)</code>
Function Description	Get transfer bit order.

Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_SPI_LSB_FIRST</li> <li>- LL_SPI_MSB_FIRST</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 LSBFIRST LL_SPI_SetTransferBitOrder</li> </ul>

### LL\_SPI\_SetTransferDirection

Function Name	<b>_STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)</b>
Function Description	Set transfer direction mode.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>TransferDirection:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_SPI_FULL_DUPLEX</li> <li>- LL_SPI_SIMPLEX_RX</li> <li>- LL_SPI_HALF_DUPLEX_RX</li> <li>- LL_SPI_HALF_DUPLEX_TX</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 RXONLY LL_SPI_SetTransferDirection</li> <li>CR1 BIDIMODE LL_SPI_SetTransferDirection</li> <li>CR1 BIDIOE LL_SPI_SetTransferDirection</li> </ul>

### LL\_SPI\_GetTransferDirection

Function Name	<b>_STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (SPI_TypeDef * SPIx)</b>
Function Description	Get transfer direction mode.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_SPI_FULL_DUPLEX</li> <li>- LL_SPI_SIMPLEX_RX</li> <li>- LL_SPI_HALF_DUPLEX_RX</li> <li>- LL_SPI_HALF_DUPLEX_TX</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 RXONLY LL_SPI_SetTransferDirection</li> <li>CR1 BIDIMODE LL_SPI_SetTransferDirection</li> <li>CR1 BIDIOE LL_SPI_SetTransferDirection</li> </ul>

### LL\_SPI\_SetDataWidth

Function Name	<b>_STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)</b>
---------------	---

Function Description	Set frame data width.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>DataWidth:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_SPI_DATAWIDTH_8BIT</li> <li>– LL_SPI_DATAWIDTH_16BIT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DFF LL_SPI_SetDataWidth</li> </ul>

### LL\_SPI\_GetDataWidth

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetDataWidth(SPI_TypeDef * SPIx)</code>
Function Description	Get frame data width.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_SPI_DATAWIDTH_8BIT</li> <li>– LL_SPI_DATAWIDTH_16BIT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DFF LL_SPI_GetDataWidth</li> </ul>

### LL\_SPI\_EnableCRC

Function Name	<code>__STATIC_INLINE void LL_SPI_EnableCRC(SPI_TypeDef * SPIx)</code>
Function Description	Enable CRC.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CRCEN LL_SPI_EnableCRC</li> </ul>

### LL\_SPI\_DisableCRC

Function Name	<code>__STATIC_INLINE void LL_SPI_DisableCRC(SPI_TypeDef * SPIx)</code>
Function Description	Disable CRC.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit should be written only when SPI is disabled (SPE = 0)</li> </ul>

for correct operation.

- Reference Manual to LL API cross reference:
- CR1 CRCEN LL\_SPI\_DisableCRC

### **LL\_SPI\_IsEnabledCRC**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)</code>
Function Description	Check if CRC is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit should be written only when SPI is disabled (SPE = 0) for correct operation.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CRCEN LL_SPI_IsEnabledCRC</li> </ul>

### **LL\_SPI\_SetCRCNext**

Function Name	<code>__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)</code>
Function Description	Set CRCNext to transfer CRC on the line.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit has to be written as soon as the last data is written in the SPIx_DR register.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CRCNEXT LL_SPI_SetCRCNext</li> </ul>

### **LL\_SPI\_SetCRCPolynomial**

Function Name	<code>__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)</code>
Function Description	Set polynomial for CRC calculation.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>CRCPoly:</b> This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CRCPR CRCPOLY LL_SPI_SetCRCPolynomial</li> </ul>

**LL\_SPI\_GetCRCPolynomial**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (SPI_TypeDef * SPIx)</code>
Function Description	Get polynomial for CRC calculation.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CRCPR CRCPOLY LL_SPI_GetCRCPolynomial</li> </ul>

**LL\_SPI\_GetRxCRC**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)</code>
Function Description	Get Rx CRC.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RXCRCR RXCRC LL_SPI_GetRxCRC</li> </ul>

**LL\_SPI\_GetTxCRC**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)</code>
Function Description	Get Tx CRC.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TXCRCR TXCRC LL_SPI_GetTxCRC</li> </ul>

**LL\_SPI\_SetNSSMode**

Function Name	<code>__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)</code>
Function Description	Set NSS mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> <li>• <b>NSS:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_SPI_NSS_SOFT</li> <li>- LL_SPI_NSS_HARD_INPUT</li> <li>- LL_SPI_NSS_HARD_OUTPUT</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 SSM LL_SPI_SetNSSMode</li> <li>CR2 SSOE LL_SPI_SetNSSMode</li> </ul>

### LL\_SPI\_GetNSSMode

Function Name	<code>_STATIC_INLINE uint32_t LL_SPI_GetNSSMode(SPI_TypeDef * SPIx)</code>
Function Description	Get NSS mode.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_SPI_NSS_SOFT</li> <li>- LL_SPI_NSS_HARD_INPUT</li> <li>- LL_SPI_NSS_HARD_OUTPUT</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 SSM LL_SPI_GetNSSMode</li> <li>CR2 SSOE LL_SPI_GetNSSMode</li> </ul>

### LL\_SPI\_IsActiveFlag\_RXNE

Function Name	<code>_STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE(SPI_TypeDef * SPIx)</code>
Function Description	Check if Rx buffer is not empty.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR RXNE LL_SPI_IsActiveFlag_RXNE</li> </ul>

### LL\_SPI\_IsActiveFlag\_TXE

Function Name	<code>_STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE(SPI_TypeDef * SPIx)</code>
Function Description	Check if Tx buffer is empty.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SR TXE LL_SPI_IsActiveFlag_TXE</li> </ul>

### LL\_SPI\_IsActiveFlag\_CRCERR

Function Name	<code>_STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR(SPI_TypeDef * SPIx)</code>
---------------	---

Function Description	Get CRC error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CRCERR LL_SPI_IsActiveFlag_CRCERR</li> </ul>

### LL\_SPI\_IsActiveFlag\_MODF

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF(SPI_TypeDef * SPIx)</code>
Function Description	Get mode fault error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR MODF LL_SPI_IsActiveFlag_MODF</li> </ul>

### LL\_SPI\_IsActiveFlag\_OVR

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR(SPI_TypeDef * SPIx)</code>
Function Description	Get overrun error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR OVR LL_SPI_IsActiveFlag_OVR</li> </ul>

### LL\_SPI\_IsActiveFlag\_BSY

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_BSY(SPI_TypeDef * SPIx)</code>
Function Description	Get busy flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR BSY LL_SPI_IsActiveFlag_BSY</li> </ul>

**LL\_SPI\_IsActiveFlag\_FRE**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (SPI_TypeDef * SPIx)</code>
Function Description	Get frame format error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR FRE LL_SPI_IsActiveFlag_FRE</li> </ul>

**LL\_SPI\_ClearFlag\_CRCERR**

Function Name	<code>__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)</code>
Function Description	Clear CRC error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CRCERR LL_SPI_ClearFlag_CRCERR</li> </ul>

**LL\_SPI\_ClearFlag\_MODF**

Function Name	<code>__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)</code>
Function Description	Clear mode fault error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Clearing this flag is done by a read access to the SPIx_SR register followed by a write access to the SPIx_CR1 register</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR MODF LL_SPI_ClearFlag_MODF</li> </ul>

**LL\_SPI\_ClearFlag\_OVR**

Function Name	<code>__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)</code>
Function Description	Clear overrun error flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Clearing this flag is done by a read access to the SPIx_DR register followed by a read access to the SPIx_SR register</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• SR OVR LL_SPI_ClearFlag_OVR</li> </ul>

LL API cross  
reference:

### LL\_SPI\_ClearFlag\_FRE

Function Name	<code>_STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef *     SPIx)</code>
Function Description	Clear frame format error flag.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Notes	<ul style="list-style-type: none"><li>Clearing this flag is done by reading SPIx_SR register</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>SR FRE LL_SPI_ClearFlag_FRE</li></ul>

### LL\_SPI\_EnableIT\_ERR

Function Name	<code>_STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef *     SPIx)</code>
Function Description	Enable error interrupt.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Notes	<ul style="list-style-type: none"><li>This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>CR2 ERRIE LL_SPI_EnableIT_ERR</li></ul>

### LL\_SPI\_EnableIT\_RXNE

Function Name	<code>_STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef *     SPIx)</code>
Function Description	Enable Rx buffer not empty interrupt.
Parameters	<ul style="list-style-type: none"><li><b>SPIx:</b> SPI Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>

### LL\_SPI\_EnableIT\_TXE

Function Name	<code>_STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef *     SPIx)</code>
Function Description	Enable Tx buffer empty interrupt.

---

Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 TXEIE LL_SPI_EnableIT_TXE</li> </ul>

### LL\_SPI\_DisableIT\_ERR

Function Name	<code>_STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef *           SPIx)</code>
Function Description	Disable error interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 ERRIE LL_SPI_DisableIT_ERR</li> </ul>

### LL\_SPI\_DisableIT\_RXNE

Function Name	<code>_STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef *           SPIx)</code>
Function Description	Disable Rx buffer not empty interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 RXNEIE LL_SPI_DisableIT_RXNE</li> </ul>

### LL\_SPI\_DisableIT\_TXE

Function Name	<code>_STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef *           SPIx)</code>
Function Description	Disable Tx buffer empty interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 TXEIE LL_SPI_DisableIT_TXE</li> </ul>

### LL\_SPI\_IsEnabledIT\_ERR

Function Name	<code>_STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (SPI_TypeDef * SPIx)</code>
---------------	--

Function Description	Check if error interrupt is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ERRIE LL_SPI_IsEnabledIT_ERR</li> </ul>

### **LL\_SPI\_IsEnabledIT\_RXNE**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXNE(SPI_TypeDef * SPIx)</code>
Function Description	Check if Rx buffer not empty interrupt is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RXNEIE LL_SPI_IsEnabledIT_RXNE</li> </ul>

### **LL\_SPI\_IsEnabledIT\_TXE**

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXE(SPI_TypeDef * SPIx)</code>
Function Description	Check if Tx buffer empty interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TXEIE LL_SPI_IsEnabledIT_TXE</li> </ul>

### **LL\_SPI\_EnableDMAReq\_RX**

Function Name	<code>__STATIC_INLINE void LL_SPI_EnableDMAReq_RX(SPI_TypeDef * SPIx)</code>
Function Description	Enable DMA Rx.
Parameters	<ul style="list-style-type: none"> <li>• <b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RXDMAEN LL_SPI_EnableDMAReq_RX</li> </ul>

### **LL\_SPI\_DisableDMAReq\_RX**

Function Name	<code>__STATIC_INLINE void LL_SPI_DisableDMAReq_RX(SPI_TypeDef * SPIx)</code>
Function Description	Disable DMA Rx.

---

Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 RXDMAEN LL_SPI_DisableDMAReq_RX</li> </ul>

### LL\_SPI\_IsEnabledDMAReq\_RX

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX(SPI_TypeDef * SPIx)</code>
Function Description	Check if DMA Rx is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 RXDMAEN LL_SPI_IsEnabledDMAReq_RX</li> </ul>

### LL\_SPI\_EnableDMAReq\_TX

Function Name	<code>__STATIC_INLINE void LL_SPI_EnableDMAReq_TX(SPI_TypeDef * SPIx)</code>
Function Description	Enable DMA Tx.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 TXDMAEN LL_SPI_EnableDMAReq_TX</li> </ul>

### LL\_SPI\_DisableDMAReq\_TX

Function Name	<code>__STATIC_INLINE void LL_SPI_DisableDMAReq_TX(SPI_TypeDef * SPIx)</code>
Function Description	Disable DMA Tx.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 TXDMAEN LL_SPI_DisableDMAReq_TX</li> </ul>

### LL\_SPI\_IsEnabledDMAReq\_TX

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX(SPI_TypeDef * SPIx)</code>
Function Description	Check if DMA Tx is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX</li> </ul>

### LL\_SPI\_DMA\_GetRegAddr

Function Name	<code>__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr(SPI_TypeDef * SPIx)</code>
Function Description	Get the data register address used for DMA transfer.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Address:</b> of data register</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_SPI_DMA_GetRegAddr</li> </ul>

### LL\_SPI\_ReceiveData8

Function Name	<code>__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)</code>
Function Description	Read 8-Bits in the data register.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>RxData:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_SPI_ReceiveData8</li> </ul>

### LL\_SPI\_ReceiveData16

Function Name	<code>__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)</code>
Function Description	Read 16-Bits in the data register.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>RxData:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_SPI_ReceiveData16</li> </ul>

### LL\_SPI\_TransmitData8

Function Name	<code>__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)</code>
Function Description	Write 8-Bits in the data register.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>TxDATA:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_SPI_TransmitData8</li> </ul>

### **LL\_SPI\_TransmitData16**

Function Name	<b>_STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxDATA)</b>
Function Description	Write 16-Bits in the data register.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>TxDATA:</b> Value between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>DR DR LL_SPI_TransmitData16</li> </ul>

### **LL\_SPI\_DelInit**

Function Name	<b>ErrorStatus LL_SPI_DelInit (SPI_TypeDef * SPIx)</b>
Function Description	De-initialize the SPI registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>SUCCESS: SPI registers are de-initialized</li> <li>ERROR: SPI registers are not de-initialized</li> </ul> </li> </ul>

### **LL\_SPI\_Init**

Function Name	<b>ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)</b>
Function Description	Initialize the SPI registers according to the specified parameters in SPI_InitStruct.
Parameters	<ul style="list-style-type: none"> <li><b>SPIx:</b> SPI Instance</li> <li><b>SPI_InitStruct:</b> pointer to a LL_SPI_InitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value. (Return always SUCCESS)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> </ul>

### **LL\_SPI\_StructInit**

Function Name	<b>void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)</b>
---------------	---

Function Description	Set each LL_SPI_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li><b>SPI_InitStruct:</b> pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>

## 72.3 SPI Firmware driver defines

### 72.3.1 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
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
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
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
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_UDR	Underrun 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
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_UDR	Underrun 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_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
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
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

---

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
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
LL_SPI_POLARITY_LOW	Clock to 0 when idle
LL_SPI_POLARITY_HIGH	Clock to 1 when idle

**Serial Protocol**

LL_SPI_PROTOCOL_MOTOROLA	Motorola mode. Used as default value
LL_SPI_PROTOCOL_TI	TI mode
LL_SPI_PROTOCOL_MOTOROLA	Motorola mode. Used as default value
LL_SPI_PROTOCOL_TI	TI mode

**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
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

[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

## 73 LL SYSTEM Generic Driver

### 73.1 SYSTEM Firmware driver API description

#### 73.1.1 Detailed description of functions

##### **LL\_SYSCFG\_SetRemapMemory**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_SetRemapMemory(   uint32_t Memory)</code>
Function Description	Set memory mapping at address 0x00000000.
Parameters	<ul style="list-style-type: none"> <li>• <b>Memory:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_SYSCFG_REMAP_FLASH</code></li> <li>- <code>LL_SYSCFG_REMAP_SYSTEMFLASH</code></li> <li>- <code>LL_SYSCFG_REMAP_SRAM</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR1 MEM_MODE <code>LL_SYSCFG_SetRemapMemory</code></li> </ul>

##### **LL\_SYSCFG\_GetRemapMemory**

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_GetRemapMemory(   void )</code>
Function Description	Get memory mapping at address 0x00000000.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_SYSCFG_REMAP_FLASH</code></li> <li>- <code>LL_SYSCFG_REMAP_SYSTEMFLASH</code></li> <li>- <code>LL_SYSCFG_REMAP_SRAM</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR1 MEM_MODE <code>LL_SYSCFG_GetRemapMemory</code></li> </ul>

##### **LL\_SYSCFG\_SetFlashBankMode**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_SetFlashBankMode(   uint32_t Bank)</code>
Function Description	Select Flash bank mode (Bank flashed at 0x08000000)
Parameters	<ul style="list-style-type: none"> <li>• <b>Bank:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_SYSCFG_BANKMODE_BANK1</code></li> <li>- <code>LL_SYSCFG_BANKMODE_BANK2</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR1 UFB <code>LL_SYSCFG_SetFlashBankMode</code></li> </ul>

**LL\_SYSCFG\_GetFlashBankMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashBankMode(void)</code>
Function Description	Get Flash bank mode (Bank flashed at 0x08000000)
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_SYSCFG_BANKMODE_BANK1</code></li> <li>- <code>LL_SYSCFG_BANKMODE_BANK2</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>SYSCFG_CFGR1 UFB LL_SYSCFG_GetFlashBankMode</code></li> </ul>

**LL\_SYSCFG\_GetBootMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_GetBootMode(void)</code>
Function Description	Get Boot mode selected by the boot pins status bits.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>LL_SYSCFG_BOOTMODE_FLASH</code></li> <li>- <code>LL_SYSCFG_BOOTMODE_SYSTEMFLASH</code></li> <li>- <code>LL_SYSCFG_BOOTMODE_SRAM</code></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• It indicates the boot mode selected by the boot pins. Bit 9 corresponds to the complement of nBOOT1 bit in the FLASH_OPTR register. Its value is defined in the option bytes. Bit 8 corresponds to the value sampled on the BOOT0 pin.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>SYSCFG_CFGR1 BOOT_MODE LL_SYSCFG_GetBootMode</code></li> </ul>

**LL\_SYSCFG\_EnableFirewall**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_EnableFirewall(void)</code>
Function Description	Firewall protection enabled.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>SYSCFG_CFGR2 FWDIS LL_SYSCFG_EnableFirewall</code></li> </ul>

**LL\_SYSCFG\_IsEnabledFirewall**

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledFirewall(void)</code>
Function Description	Check if Firewall protection is enabled or not.
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>SYSCFG_CFGR2 FWDIS LL_SYSCFG_IsEnabledFirewall</code></li> </ul>

**LL\_SYSCFG\_SetVLCDRailConnection**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_SetVLCDRailConnection (uint32_t IoPinConnect)</code>
Function Description	Set VLCD rail connection to optional external capacitor.
Parameters	<ul style="list-style-type: none"> <li>• <b>IoPinConnect:</b> This parameter can be a combination of the following values: (*) value not defined in all devices           <ul style="list-style-type: none"> <li>– LL_SYSCFG_CAPA_VLCD1_PB12</li> <li>– LL_SYSCFG_CAPA_VLCD1_PE11(*)</li> <li>– LL_SYSCFG_CAPA_VLCD2_PB2</li> <li>– LL_SYSCFG_CAPA_VLCD3_PB0</li> <li>– LL_SYSCFG_CAPA_VLCD3_PE12(*)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• One to three external capacitors can be connected to pads to do VLCD biasing. LCD_VLCD1 rail can be connected to PB12 or PE11(*),LCD_VLCD2 rail can be connected to PB2,LCD_VLCD3 rail can be connected to PB0 or PE12(*)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR2 CAPA <a href="#">LL_SYSCFG_SetVLCDRailConnection</a></li> </ul>

**LL\_SYSCFG\_GetVLCDRailConnection**

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_GetVLCDRailConnection (void )</code>
Function Description	Get VLCD rail connection configuration.
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be a combination of the following values: (*) value not defined in all devices           <ul style="list-style-type: none"> <li>– LL_SYSCFG_CAPA_VLCD1_PB12</li> <li>– LL_SYSCFG_CAPA_VLCD1_PE11(*)</li> <li>– LL_SYSCFG_CAPA_VLCD2_PB2</li> <li>– LL_SYSCFG_CAPA_VLCD3_PB0</li> <li>– LL_SYSCFG_CAPA_VLCD3_PE12(*)</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• One to three external capacitors can be connected to pads to do VLCD biasing. LCD_VLCD1 rail can be connected to PB12 or PE11(*),LCD_VLCD2 rail can be connected to PB2,LCD_VLCD3 rail can be connected to PB0 or PE12(*)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR2 CAPA <a href="#">LL_SYSCFG_GetVLCDRailConnection</a></li> </ul>

**LL\_SYSCFG\_EnableFastModePlus**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)</code>
Function Description	Enable the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none"> <li>• <b>ConfigFastModePlus:</b> This parameter can be a combination of the following values: (*) value not defined in all devices           <ul style="list-style-type: none"> <li>– LL_SYSCFG_I2C_FASTMODEPLUS_PB6</li> </ul> </li> </ul>

- LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB7
- LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB8
- LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB9
- LL\_SYSCFG\_I2C\_FASTMODEPLUS\_I2C1
- LL\_SYSCFG\_I2C\_FASTMODEPLUS\_I2C2 (\*)
- LL\_SYSCFG\_I2C\_FASTMODEPLUS\_I2C3 (\*)

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- SYSCFG\_CFGR2\_I2C\_PBx\_FMP  
LL\_SYSCFG\_EnableFastModePlus
- SYSCFG\_CFGR2\_I2Cx\_FMP  
LL\_SYSCFG\_EnableFastModePlus

**LL\_SYSCFG\_DisableFastModePlus****Function Name**

```
_STATIC_INLINE void LL_SYSCFG_DisableFastModePlus  
(uint32_t ConfigFastModePlus)
```

**Function Description**

Disable the I2C fast mode plus driving capability.

**Parameters**

- **ConfigFastModePlus:** This parameter can be a combination of the following values: (\*) value not defined in all devices
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB6
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB7
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB8
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_PB9
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_I2C1
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_I2C2 (\*)
  - LL\_SYSCFG\_I2C\_FASTMODEPLUS\_I2C3 (\*)

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- SYSCFG\_CFGR2\_I2C\_PBx\_FMP  
LL\_SYSCFG\_DisableFastModePlus
- SYSCFG\_CFGR2\_I2Cx\_FMP  
LL\_SYSCFG\_DisableFastModePlus

**LL\_SYSCFG\_VREFINT\_SetConnection****Function Name**

```
_STATIC_INLINE void LL_SYSCFG_VREFINT_SetConnection  
(uint32_t IoPinConnect)
```

**Function Description**

Select which pad is connected to VREFINT\_ADC.

**Parameters**

- **IoPinConnect:** This parameter can be one of the following values:
  - LL\_SYSCFG\_VREFINT\_CONNECT\_NONE
  - LL\_SYSCFG\_VREFINT\_CONNECT\_IO1
  - LL\_SYSCFG\_VREFINT\_CONNECT\_IO2
  - LL\_SYSCFG\_VREFINT\_CONNECT\_IO1\_IO2

**Return values**

- **None:**

**Reference Manual to  
LL API cross  
reference:**

- SYSCFG\_CFGR3\_SEL\_VREF\_OUT  
LL\_SYSCFG\_VREFINT\_SetConnection

**LL\_SYSCFG\_VREFINT\_GetConnection**

Function Name      **`_STATIC_INLINE uint32_t  
LL_SYSCFG_VREFINT_GetConnection (void )`**

Function Description      Get pad connection to VREFINT\_ADC.

Return values     
 

- **Returned:** value can be one of the following values:
  - LL\_SYSCFG\_VREFINT\_CONNECT\_NONE
  - LL\_SYSCFG\_VREFINT\_CONNECT\_IO1
  - LL\_SYSCFG\_VREFINT\_CONNECT\_IO2
  - LL\_SYSCFG\_VREFINT\_CONNECT\_IO1\_IO2

Reference Manual to  
LL API cross  
reference:  

- SYSCFG\_CFGR3 SEL\_VREF\_OUT  
`LL_SYSCFG_VREFINT_GetConnection`

**LL\_SYSCFG\_VREFINT\_EnableADC**

Function Name      **`_STATIC_INLINE void LL_SYSCFG_VREFINT_EnableADC  
(void )`**

Function Description      Buffer used to generate VREFINT reference for ADC enable.

Return values     
 

- **None:**

Notes     
 

- The VrefInit buffer to ADC through internal path is also enabled using function  
`LL_ADC_SetCommonPathInternalCh()` with parameter  
`LL_ADC_PATH_INTERNAL_VREFINT`

Reference Manual to  
LL API cross  
reference:  

- SYSCFG\_CFGR3 ENBUF\_VREFINT\_ADC  
`LL_SYSCFG_VREFINT_EnableADC`

**LL\_SYSCFG\_VREFINT\_DisableADC**

Function Name      **`_STATIC_INLINE void LL_SYSCFG_VREFINT_DisableADC  
(void )`**

Function Description      Buffer used to generate VREFINT reference for ADC disable.

Return values     
 

- **None:**

Reference Manual to  
LL API cross  
reference:  

- SYSCFG\_CFGR3 ENBUF\_VREFINT\_ADC  
`LL_SYSCFG_VREFINT_DisableADC`

**LL\_SYSCFG\_TEMPSENSOR\_Enable**

Function Name      **`_STATIC_INLINE void LL_SYSCFG_TEMPSENSOR_Enable  
(void )`**

Function Description      Buffer used to generate temperature sensor reference for ADC enable.

Return values     
 

- **None:**

Reference Manual to  
LL API cross  
reference:  

- SYSCFG\_CFGR3 ENBUF\_SENSOR\_ADC  
`LL_SYSCFG_TEMPSENSOR_Enable`

**LL\_SYSCFG\_TEMPSENSOR\_Disable**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_TEMPSENSOR_Disable(void)</code>
Function Description	Buffer used to generate temperature sensor reference for ADC disable.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR3_ENBUF_SENSOR_ADC <code>LL_SYSCFG_TEMPSENSOR_Disable</code></li> </ul>

**LL\_SYSCFG\_VREFINT\_EnableCOMP**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_VREFINT_EnableCOMP(void)</code>
Function Description	Buffer used to generate VREFINT reference for comparator enable.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR3_ENBUF_VREFINT_COMP <code>LL_SYSCFG_VREFINT_EnableCOMP</code></li> </ul>

**LL\_SYSCFG\_VREFINT\_DisableCOMP**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_VREFINT_DisableCOMP(void)</code>
Function Description	Buffer used to generate VREFINT reference for comparator disable.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR3_ENBUF_VREFINT_COMP <code>LL_SYSCFG_VREFINT_DisableCOMP</code></li> </ul>

**LL\_SYSCFG\_VREFINT\_EnableHSI48**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_VREFINT_EnableHSI48(void)</code>
Function Description	Buffer used to generate VREFINT reference for HSI48 oscillator enable.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_CFGR3_ENREF_HSI48 <code>LL_SYSCFG_VREFINT_EnableHSI48</code></li> </ul>

**LL\_SYSCFG\_VREFINT\_DisableHSI48**

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_VREFINT_DisableHSI48(void)</code>
---------------	--

Function Description	Buffer used to generate VREFINT reference for HSI48 oscillator disable.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SYSCFG_CFGR3 ENREF_HSI48</li> <li>LL_SYSCFG_VREFINT_DisableHSI48</li> </ul>

### LL\_SYSCFG\_VREFINT\_IsReady

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_VREFINT_IsReady(void)</code>
Function Description	Check if VREFINT is ready or not.
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When set, it indicates that VREFINT is available for BOR, PVD and LCD</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SYSCFG_CFGR3 VREFINT_RDYF</li> <li>LL_SYSCFG_VREFINT_IsReady</li> </ul>

### LL\_SYSCFG\_VREFINT\_Lock

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_VREFINT_Lock(void)</code>
Function Description	Lock the whole content of SYSCFG_CFGR3 register.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>After SYSCFG_CFGR3 register lock, only read access available. Only system hardware reset unlocks SYSCFG_CFGR3 register.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SYSCFG_CFGR3 REF_LOCK</li> <li>LL_SYSCFG_VREFINT_Lock</li> </ul>

### LL\_SYSCFG\_VREFINT\_IsLocked

Function Name	<code>__STATIC_INLINE uint32_t LL_SYSCFG_VREFINT_IsLocked(void)</code>
Function Description	Check if SYSCFG_CFGR3 register is locked (only read access) or not.
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When set, it indicates that SYSCFG_CFGR3 register is locked, only read access available</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SYSCFG_CFGR3 REF_LOCK</li> <li>LL_SYSCFG_VREFINT_IsLocked</li> </ul>

### LL\_SYSCFG\_SetEXTISource

Function Name	<code>__STATIC_INLINE void LL_SYSCFG_SetEXTISource(uint32_t</code>
---------------	--

**Port, uint32\_t Line)**

Function Description	Configure source input for the EXTI external interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>Port:</b> This parameter can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_SYSCFG_EXTI_PORTA</li> <li>- LL_SYSCFG_EXTI_PORTB</li> <li>- LL_SYSCFG_EXTI_PORTC</li> <li>- LL_SYSCFG_EXTI_PORTD (*)</li> <li>- LL_SYSCFG_EXTI_PORTE (*)</li> <li>- LL_SYSCFG_EXTI_PORTH (*)</li> </ul> </li> <li>• <b>Line:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_SYSCFG_EXTI_LINE0</li> <li>- LL_SYSCFG_EXTI_LINE1</li> <li>- LL_SYSCFG_EXTI_LINE2</li> <li>- LL_SYSCFG_EXTI_LINE3</li> <li>- LL_SYSCFG_EXTI_LINE4</li> <li>- LL_SYSCFG_EXTI_LINE5</li> <li>- LL_SYSCFG_EXTI_LINE6</li> <li>- LL_SYSCFG_EXTI_LINE7</li> <li>- LL_SYSCFG_EXTI_LINE8</li> <li>- LL_SYSCFG_EXTI_LINE9</li> <li>- LL_SYSCFG_EXTI_LINE10</li> <li>- LL_SYSCFG_EXTI_LINE11</li> <li>- LL_SYSCFG_EXTI_LINE12</li> <li>- LL_SYSCFG_EXTI_LINE13</li> <li>- LL_SYSCFG_EXTI_LINE14</li> <li>- LL_SYSCFG_EXTI_LINE15</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_EXTICR1 EXTI0 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR1 EXTI1 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR1 EXTI2 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR1 EXTI3 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI4 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI5 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI6 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI7 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI8 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI9 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI10 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI11 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI12 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI13 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI14 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI15 LL_SYSCFG_SetEXTISource</li> </ul>

**LL\_SYSCFG\_GetEXTISource**

Function Name	<code>_STATIC_INLINE uint32_t LL_SYSCFG_GetEXTISource (uint32_t Line)</code>
Function Description	Get the configured defined for specific EXTI Line.
Parameters	<ul style="list-style-type: none"> <li><b>Line:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_SYSCFG_EXTI_LINE0</li> <li>- LL_SYSCFG_EXTI_LINE1</li> <li>- LL_SYSCFG_EXTI_LINE2</li> <li>- LL_SYSCFG_EXTI_LINE3</li> <li>- LL_SYSCFG_EXTI_LINE4</li> <li>- LL_SYSCFG_EXTI_LINE5</li> <li>- LL_SYSCFG_EXTI_LINE6</li> <li>- LL_SYSCFG_EXTI_LINE7</li> <li>- LL_SYSCFG_EXTI_LINE8</li> <li>- LL_SYSCFG_EXTI_LINE9</li> <li>- LL_SYSCFG_EXTI_LINE10</li> <li>- LL_SYSCFG_EXTI_LINE11</li> <li>- LL_SYSCFG_EXTI_LINE12</li> <li>- LL_SYSCFG_EXTI_LINE13</li> <li>- LL_SYSCFG_EXTI_LINE14</li> <li>- LL_SYSCFG_EXTI_LINE15</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_SYSCFG_EXTI_PORTA</li> <li>- LL_SYSCFG_EXTI_PORTB</li> <li>- LL_SYSCFG_EXTI_PORTC</li> <li>- LL_SYSCFG_EXTI_PORTD (*)</li> <li>- LL_SYSCFG_EXTI_PORTE (*)</li> <li>- LL_SYSCFG_EXTI_PORTH (*)</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SYSCFG_EXTICR1 EXTI0 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR1 EXTI1 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR1 EXTI2 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR1 EXTI3 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI4 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI5 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI6 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR2 EXTI7 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI8 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI9 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI10 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR3 EXTI11 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI12 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI13 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI14 LL_SYSCFG_SetEXTISource</li> <li>• SYSCFG_EXTICR4 EXTI15 LL_SYSCFG_SetEXTISource</li> </ul>

**LL\_DBGMCU\_GetDeviceID**

Function Name	<code>__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void )</code>
Function Description	Return the device identifier.
Return values	<ul style="list-style-type: none"> <li>• <b>Values:</b> between Min_Data=0x00 and Max_Data=0x7FF (ex: L053 -&gt; 0x417, L073 -&gt; 0x447)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID</code></li> </ul>

**LL\_DBGMCU\_GetRevisionID**

Function Name	<code>__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void )</code>
Function Description	Return the device revision identifier.
Return values	<ul style="list-style-type: none"> <li>• <b>Values:</b> between Min_Data=0x00 and Max_Data=0xFFFF</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This field indicates the revision of the device.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID</code></li> </ul>

**LL\_DBGMCU\_EnableDBGSleepMode**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_EnableDBGSleepMode (void )</code>
Function Description	Enable the Debug Module during SLEEP mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>DBGMCU_CR DBG_SLEEP</code></li> <li>• <code>LL_DBGMCU_EnableDBGSleepMode</code></li> </ul>

**LL\_DBGMCU\_DisableDBGSleepMode**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_DisableDBGSleepMode (void )</code>
Function Description	Disable the Debug Module during SLEEP mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• <code>DBGMCU_CR DBG_SLEEP</code></li> <li>• <code>LL_DBGMCU_DisableDBGSleepMode</code></li> </ul>

**LL\_DBGMCU\_EnableDBGStopMode**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void )</code>
Function Description	Enable the Debug Module during STOP mode.

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode</li> </ul>

**LL\_DBGMCU\_DisableDBGStopMode**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void )</code>
Function Description	Disable the Debug Module during STOP mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode</li> </ul>

**LL\_DBGMCU\_EnableDBGStandbyMode**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void )</code>
Function Description	Enable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode</li> </ul>

**LL\_DBGMCU\_DisableDBGStandbyMode**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void )</code>
Function Description	Disable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode</li> </ul>

**LL\_DBGMCU\_ABP1\_GRP1\_FreezePeriph**

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_ABP1_GRP1_FreezePeriph (uint32_t Periph)</code>
Function Description	Freeze APB1 peripherals (group1 peripherals)
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_DBGMCU_ABP1_GRP1_TIM2_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_TIM3_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_TIM6_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_TIM7_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_RTC_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_WWDG_STOP</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_DBGMCU_ABP1_GRP1_IWDG_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_I2C1_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_I2C2_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_I2C3_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_LPTIM1_STOP</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB1FZ DBG_TIM2_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_TIM3_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_TIM6_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_TIM7_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_RTC_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_WWDG_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_IWDG_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_I2C1_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_I2C2_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_I2C3_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> <li>• APB1FZ DBG_LPTIMER_STOP LL_DBGMCU_ABP1_GRP1_FreezePeriph</li> </ul>

### LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph

Function Name	<code>__STATIC_INLINE void LL_DBGMCU_ABP1_GRP1_UnFreezePeriph (uint32_t Periphs)</code>
Function Description	Unfreeze APB1 peripherals (group1 peripherals)
Parameters	<ul style="list-style-type: none"> <li>• <b>Periphs:</b> This parameter can be a combination of the following values: (*) value not defined in all devices <ul style="list-style-type: none"> <li>- LL_DBGMCU_ABP1_GRP1_TIM2_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_TIM3_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_TIM6_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_TIM7_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_RTC_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_WWDG_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_IWDG_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_I2C1_STOP</li> <li>- LL_DBGMCU_ABP1_GRP1_I2C2_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_I2C3_STOP (*)</li> <li>- LL_DBGMCU_ABP1_GRP1_LPTIM1_STOP</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• APB1FZ DBG_TIM2_STOP</li> </ul>

- LL API cross reference:
- LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_TIM3\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_TIM6\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_TIM7\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_RTC\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_WWDG\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_IWDG\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_I2C1\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_I2C2\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_I2C3\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph
  - APB1FZ DBG\_LPTIMER\_STOP
  - LL\_DBGMCU\_ABP1\_GRP1\_UnFreezePeriph

### **LL\_DBGMCU\_ABP2\_GRP1\_FreezePeriph**

Function Name	<b><code>__STATIC_INLINE void LL_DBGMCU_ABP2_GRP1_FreezePeriph (uint32_t Periph)</code></b>
Function Description	Freeze APB2 peripherals.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices           <ul style="list-style-type: none"> <li>- LL_DBGMCU_ABP2_GRP1_TIM22_STOP (*)</li> <li>- LL_DBGMCU_ABP2_GRP1_TIM21_STOP</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• APB2FZ DBG_TIM22_STOP</li> <li>• LL_DBGMCU_ABP2_GRP1_FreezePeriph</li> <li>• APB2FZ DBG_TIM21_STOP</li> <li>• LL_DBGMCU_ABP2_GRP1_FreezePeriph</li> </ul>

### **LL\_DBGMCU\_ABP2\_GRP1\_UnFreezePeriph**

Function Name	<b><code>__STATIC_INLINE void LL_DBGMCU_ABP2_GRP1_UnFreezePeriph (uint32_t Periph)</code></b>
Function Description	Unfreeze APB2 peripherals.
Parameters	<ul style="list-style-type: none"> <li>• <b>Periph:</b> This parameter can be a combination of the following values: (*) value not defined in all devices           <ul style="list-style-type: none"> <li>- LL_DBGMCU_ABP2_GRP1_TIM22_STOP (*)</li> <li>- LL_DBGMCU_ABP2_GRP1_TIM21_STOP</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• APB2FZ DBG_TIM22_STOP</li> <li>• LL_DBGMCU_ABP2_GRP1_UnFreezePeriph</li> </ul>

- reference:
- APB2FZ DBG\_TIM21\_STOP  
LL\_DBGMCU\_ABP2\_GRP1\_UnFreezePeriph

### **LL\_FLASH\_SetLatency**

Function Name	<b><code>_STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)</code></b>
Function Description	Set FLASH Latency.
Parameters	<ul style="list-style-type: none"> <li><b>Latency:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_FLASH_LATENCY_0</li> <li>- LL_FLASH_LATENCY_1</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR LATENCY LL_FLASH_SetLatency</li> </ul>

### **LL\_FLASH\_GetLatency**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_FLASH_GetLatency (void )</code></b>
Function Description	Get FLASH Latency.
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_FLASH_LATENCY_0</li> <li>- LL_FLASH_LATENCY_1</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR LATENCY LL_FLASH_GetLatency</li> </ul>

### **LL\_FLASH\_EnablePrefetch**

Function Name	<b><code>_STATIC_INLINE void LL_FLASH_EnablePrefetch (void )</code></b>
Function Description	Enable Prefetch.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR PRFTEN LL_FLASH_EnablePrefetch</li> </ul>

### **LL\_FLASH\_DisablePrefetch**

Function Name	<b><code>_STATIC_INLINE void LL_FLASH_DisablePrefetch (void )</code></b>
Function Description	Disable Prefetch.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR PRFTEN LL_FLASH_DisablePrefetch</li> </ul>

**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 PRFTEN LL\_FLASH\_IsPrefetchEnabled

**LL\_FLASH\_EnableRunPowerDown**

Function Name      **`_STATIC_INLINE void LL_FLASH_EnableRunPowerDown(void)`**

Function Description      Enable Flash Power-down mode during run mode or Low-power run mode.

Return values      • **None:**

Notes      • Flash memory can be put in power-down mode only when the code is executed from RAM  
 • Flash must not be accessed when power down is enabled  
 • Flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to  
LL API cross  
reference:

- FLASH\_ACR RUN\_PD LL\_FLASH\_EnableRunPowerDown
- FLASH\_PDKEYR PDKEY1  
LL\_FLASH\_EnableRunPowerDown
- FLASH\_PDKEYR PDKEY2  
LL\_FLASH\_EnableRunPowerDown

**LL\_FLASH\_DisableRunPowerDown**

Function Name      **`_STATIC_INLINE void LL_FLASH_DisableRunPowerDown(void)`**

Function Description      Disable Flash Power-down mode during run mode or Low-power run mode.

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:

- FLASH\_ACR RUN\_PD LL\_FLASH\_DisableRunPowerDown
- FLASH\_PDKEYR PDKEY1  
LL\_FLASH\_DisableRunPowerDown
- FLASH\_PDKEYR PDKEY2  
LL\_FLASH\_DisableRunPowerDown

**LL\_FLASH\_EnableSleepPowerDown**

Function Name      **`_STATIC_INLINE void LL_FLASH_EnableSleepPowerDown(void)`**

Function Description      Enable Flash Power-down mode during Sleep or Low-power sleep mode.

Return values      • **None:**

---

Notes	<ul style="list-style-type: none"> <li>Flash must not be put in power-down while a program or an erase operation is on-going</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR SLEEP_PD</li> <li>LL_FLASH_EnableSleepPowerDown</li> </ul>

### LL\_FLASH\_DisableSleepPowerDown

Function Name	<code>__STATIC_INLINE void LL_FLASH_DisableSleepPowerDown(void)</code>
Function Description	Disable Flash Power-down mode during Sleep or Low-power sleep mode.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR SLEEP_PD</li> <li>LL_FLASH_DisableSleepPowerDown</li> </ul>

### LL\_FLASH\_EnableBuffers

Function Name	<code>__STATIC_INLINE void LL_FLASH_EnableBuffers(void)</code>
Function Description	Enable buffers used as a cache during read access.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR DISAB_BUF LL_FLASH_EnableBuffers</li> </ul>

### LL\_FLASH\_DisableBuffers

Function Name	<code>__STATIC_INLINE void LL_FLASH_DisableBuffers(void)</code>
Function Description	Disable buffers used as a cache during read access.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When disabled, every read will access the NVM even for an address already read (for example, the previous address).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>FLASH_ACR DISAB_BUF LL_FLASH_DisableBuffers</li> </ul>

### LL\_FLASH\_EnablePreRead

Function Name	<code>__STATIC_INLINE void LL_FLASH_EnablePreRead(void)</code>
Function Description	Enable pre-read.
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When enabled, the memory interface stores the last address read as data and tries to read the next one when no other read or write or prefetch operation is ongoing. It is automatically disabled every time the buffers are disabled.</li> </ul>

- Reference Manual to • FLASH\_ACR PRE\_READ LL\_FLASH\_EnablePreRead  
LL API cross reference:

### **LL\_FLASH\_DisablePreRead**

Function Name	<code>__STATIC_INLINE void LL_FLASH_DisablePreRead (void)</code>
Function Description	Disable pre-read.
Return values	• <b>None:</b>
Reference Manual to LL API cross reference:	• FLASH_ACR PRE_READ LL_FLASH_DisablePreRead

## **73.2 SYSTEM Firmware driver defines**

### **73.2.1 SYSTEM**

#### ***DBGMCU ABP1 GRP1 STOP IP***

<code>LL_DBGMCU_ABP1_GRP1_TIM2_STOP</code>	TIM2 counter stopped when core is halted
<code>LL_DBGMCU_ABP1_GRP1_TIM3_STOP</code>	TIM3 counter stopped when core is halted
<code>LL_DBGMCU_ABP1_GRP1_TIM6_STOP</code>	TIM6 counter stopped when core is halted
<code>LL_DBGMCU_ABP1_GRP1_TIM7_STOP</code>	TIM7 counter stopped when core is halted
<code>LL_DBGMCU_ABP1_GRP1_RTC_STOP</code>	RTC Calendar frozen when core is halted
<code>LL_DBGMCU_ABP1_GRP1_WWDG_STOP</code>	Debug Window Watchdog stopped when Core is halted
<code>LL_DBGMCU_ABP1_GRP1_IWDG_STOP</code>	Debug Independent Watchdog stopped when Core is halted
<code>LL_DBGMCU_ABP1_GRP1_I2C1_STOP</code>	I2C1 SMBUS timeout mode stopped when Core is halted
<code>LL_DBGMCU_ABP1_GRP1_I2C2_STOP</code>	I2C2 SMBUS timeout mode stopped when Core is halted
<code>LL_DBGMCU_ABP1_GRP1_I2C3_STOP</code>	I2C3 SMBUS timeout mode stopped when Core is halted
<code>LL_DBGMCU_ABP1_GRP1_LPTIM1_STOP</code>	LPTIM1 counter stopped when core is halted

#### ***DBGMCU ABP2 GRP1 STOP IP***

<code>LL_DBGMCU_ABP2_GRP1_TIM22_STOP</code>	TIM22 counter stopped when core is halted
<code>LL_DBGMCU_ABP2_GRP1_TIM21_STOP</code>	TIM21 counter stopped when core is halted

#### ***SYSCFG Bank Mode***

<code>LL_SYSCFG_BANKMODE_BANK1</code>	Flash Bank1 mapped at 0x08000000 (and aliased at 0x00000000), Flash Bank2 mapped at 0x08018000 (and aliased at 0x00018000), Data EEPROM Bank1 mapped at 0x08080000 (and aliased at 0x00080000), Data EEPROM Bank2 mapped at
---------------------------------------	---

0x08080C00 (and aliased at 0x00080C00)

`LL_SYSCFG_BANKMODE_BANK2` Flash Bank2 mapped at 0x08000000 (and aliased at 0x00000000), Flash Bank1 mapped at 0x08018000 (and aliased at 0x00018000), Data EEPROM Bank2 mapped at 0x08080000 (and aliased at 0x00080000), Data EEPROM Bank1 mapped at 0x08080C00 (and aliased at 0x00080C00)

#### **SYSCFG Boot Mode**

<code>LL_SYSCFG_BOOTMODE_FLASH</code>	Main Flash memory boot mode
<code>LL_SYSCFG_BOOTMODE_SYSTEMFLASH</code>	System Flash memory boot mode
<code>LL_SYSCFG_BOOTMODE_SRAM</code>	SRAM boot mode

#### **SYSCFG VLCD Rail Connection**

<code>LL_SYSCFG_CAPA_VLCD2_PB2</code>	Connect PB2 pin to LCD_VLCD2 rails supply voltage
<code>LL_SYSCFG_CAPA_VLCD1_PB12</code>	Connect PB12 pin to LCD_VLCD1 rails supply voltage
<code>LL_SYSCFG_CAPA_VLCD3_PB0</code>	Connect PB0 pin to LCD_VLCD3 rails supply voltage
<code>LL_SYSCFG_CAPA_VLCD1_PE11</code>	Connect PE11 pin to LCD_VLCD1 rails supply voltage
<code>LL_SYSCFG_CAPA_VLCD3_PE12</code>	Connect PE12 pin to LCD_VLCD3 rails supply voltage

#### **SYSCFG EXTI Line**

<code>LL_SYSCFG_EXTI_LINE0</code>	<code>EXTI_POSITION_0   EXTICR[0]</code>
<code>LL_SYSCFG_EXTI_LINE1</code>	<code>EXTI_POSITION_4   EXTICR[0]</code>
<code>LL_SYSCFG_EXTI_LINE2</code>	<code>EXTI_POSITION_8   EXTICR[0]</code>
<code>LL_SYSCFG_EXTI_LINE3</code>	<code>EXTI_POSITION_12   EXTICR[0]</code>
<code>LL_SYSCFG_EXTI_LINE4</code>	<code>EXTI_POSITION_0   EXTICR[1]</code>
<code>LL_SYSCFG_EXTI_LINE5</code>	<code>EXTI_POSITION_4   EXTICR[1]</code>
<code>LL_SYSCFG_EXTI_LINE6</code>	<code>EXTI_POSITION_8   EXTICR[1]</code>
<code>LL_SYSCFG_EXTI_LINE7</code>	<code>EXTI_POSITION_12   EXTICR[1]</code>
<code>LL_SYSCFG_EXTI_LINE8</code>	<code>EXTI_POSITION_0   EXTICR[2]</code>
<code>LL_SYSCFG_EXTI_LINE9</code>	<code>EXTI_POSITION_4   EXTICR[2]</code>
<code>LL_SYSCFG_EXTI_LINE10</code>	<code>EXTI_POSITION_8   EXTICR[2]</code>
<code>LL_SYSCFG_EXTI_LINE11</code>	<code>EXTI_POSITION_12   EXTICR[2]</code>
<code>LL_SYSCFG_EXTI_LINE12</code>	<code>EXTI_POSITION_0   EXTICR[3]</code>
<code>LL_SYSCFG_EXTI_LINE13</code>	<code>EXTI_POSITION_4   EXTICR[3]</code>
<code>LL_SYSCFG_EXTI_LINE14</code>	<code>EXTI_POSITION_8   EXTICR[3]</code>
<code>LL_SYSCFG_EXTI_LINE15</code>	<code>EXTI_POSITION_12   EXTICR[3]</code>

#### **SYSCFG EXTI Port**

<code>LL_SYSCFG_EXTI_PORTA</code>	EXTI PORT A
-----------------------------------	-------------

---

LL_SYSCFG_EXTI_PORTB	EXTI PORT B
LL_SYSCFG_EXTI_PORTC	EXTI PORT C
LL_SYSCFG_EXTI_PORTD	EXTI PORT D
LL_SYSCFG_EXTI_PORTE	EXTI PORT E
LL_SYSCFG_EXTI_PORTH	EXTI PORT H

**SYSCFG I2C FASTMODEPLUS**

LL_SYSCFG_I2C_FASTMODEPLUS_PB6	Enable Fast Mode Plus on PB6
LL_SYSCFG_I2C_FASTMODEPLUS_PB7	Enable Fast Mode Plus on PB7
LL_SYSCFG_I2C_FASTMODEPLUS_PB8	Enable Fast Mode Plus on PB8
LL_SYSCFG_I2C_FASTMODEPLUS_PB9	Enable Fast Mode Plus on PB9
LL_SYSCFG_I2C_FASTMODEPLUS_I2C1	Enable Fast Mode Plus on I2C1 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C2	Enable Fast Mode Plus on I2C2 pins
LL_SYSCFG_I2C_FASTMODEPLUS_I2C3	Enable Fast Mode Plus on I2C3 pins

**FLASH LATENCY**

LL_FLASH_LATENCY_0	FLASH Zero Latency cycle
LL_FLASH_LATENCY_1	FLASH One Latency cycle

**SYSCFG Memory Remap**

LL_SYSCFG_REMAP_FLASH	Main Flash memory mapped at 0x00000000
LL_SYSCFG_REMAP_SYSTEMFLASH	System Flash memory mapped at 0x00000000
LL_SYSCFG_REMAP_SRAM	SRAM mapped at 0x00000000

**SYSCFG VREFINT Control**

LL_SYSCFG_VREFINT_CONNECT_NONE	No pad connected to VREFINT_ADC
LL_SYSCFG_VREFINT_CONNECT_IO1	PB0 connected to VREFINT_ADC
LL_SYSCFG_VREFINT_CONNECT_IO2	PB1 connected to VREFINT_ADC
LL_SYSCFG_VREFINT_CONNECT_IO1_IO2	PB0 and PB1 connected to VREFINT_ADC

## 74 LL TIM Generic Driver

### 74.1 TIM Firmware driver registers structures

#### 74.1.1 LL\_TIM\_InitTypeDef

##### Data Fields

- *uint16\_t Prescaler*
- *uint32\_t CounterMode*
- *uint32\_t Autoreload*
- *uint32\_t ClockDivision*

##### 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 0x0000 and 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 0x0000 and 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()**.

#### 74.1.2 LL\_TIM\_OC\_InitTypeDef

##### Data Fields

- *uint32\_t OCMode*
- *uint32\_t OCState*
- *uint32\_t CompareValue*
- *uint32\_t OCPolarity*

##### 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::CompareValue***  
Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between 0x0000 and 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()**.

#### 74.1.3 LL\_TIM\_IC\_InitTypeDef

##### Data Fields

- ***uint32\_t IC\_Polarity***
- ***uint32\_t ICActiveInput***
- ***uint32\_t IC\_Prescaler***
- ***uint32\_t IC\_Filter***

##### Field Documentation

- ***uint32\_t LL\_TIM\_IC\_InitTypeDef::IC\_Polarity***  
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::IC\_Prescaler***  
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::IC\_Filter***  
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()**.

#### 74.1.4 LL\_TIM\_ENCODER\_InitTypeDef

##### Data Fields

- ***uint32\_t EncoderMode***
- ***uint32\_t IC1\_Polarity***
- ***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\_IC\_PSC**. 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\_IC\_PSC**. 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()**.

## 74.2 TIM Firmware driver API description

### 74.2.1 Detailed description of functions

#### LL\_TIM\_EnableCounter

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)</code>
Function Description	Enable timer counter.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 CEN LL_TIM_EnableCounter</li></ul>

#### LL\_TIM\_DisableCounter

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)</code>
Function Description	Disable timer counter.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 CEN LL_TIM_DisableCounter</li></ul>

#### LL\_TIM\_IsEnabledCounter

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the timer counter is enabled.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 CEN LL_TIM_IsEnabledCounter</li></ul>

#### LL\_TIM\_EnableUpdateEvent

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)</code>
Function Description	Enable update event generation.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"><li>• CR1 UDIS LL_TIM_EnableUpdateEvent</li></ul>

reference:

### **LL\_TIM\_DisableUpdateEvent**

Function Name	<b><code>__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)</code></b>
Function Description	Disable update event generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UDIS LL_TIM_DisableUpdateEvent</li> </ul>

### **LL\_TIM\_IsEnabledUpdateEvent**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (TIM_TypeDef * TIMx)</code></b>
Function Description	Indicates whether update event generation is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UDIS LL_TIM_IsEnabledUpdateEvent</li> </ul>

### **LL\_TIM\_SetUpdateSource**

Function Name	<b><code>__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)</code></b>
Function Description	Set update event source.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>UpdateSource:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_TIM_UPDATESOURCE_REGULAR</li> <li>– LL_TIM_UPDATESOURCE_COUNTER</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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</li> <li>• Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 URS LL_TIM_SetUpdateSource</li> </ul>

**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

	<ul style="list-style-type: none"> <li>• <b>CounterMode:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_COUNTERMODE_UP</li> <li>- LL_TIM_COUNTERMODE_DOWN</li> <li>- LL_TIM_COUNTERMODE_CENTER_UP</li> <li>- LL_TIM_COUNTERMODE_CENTER_DOWN</li> <li>- LL_TIM_COUNTERMODE_CENTER_UP_DOWN</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DIR LL_TIM_SetCounterMode</li> <li>• CR1 CMS LL_TIM_SetCounterMode</li> </ul>

### LL\_TIM\_GetCounterMode

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_GetCounterMode(   TIM_TypeDef * TIMx)</code>
Function Description	Get actual counter mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_COUNTERMODE_UP</li> <li>- LL_TIM_COUNTERMODE_DOWN</li> <li>- LL_TIM_COUNTERMODE_CENTER_UP</li> <li>- LL_TIM_COUNTERMODE_CENTER_DOWN</li> <li>- LL_TIM_COUNTERMODE_CENTER_UP_DOWN</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DIR LL_TIM_GetCounterMode</li> <li>• CR1 CMS LL_TIM_GetCounterMode</li> </ul>

### LL\_TIM\_EnableARRPreload

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableARRPreload(   TIM_TypeDef * TIMx)</code>
Function Description	Enable auto-reload (ARR) preload.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

Reference Manual to LL API cross reference:

- CR1 ARPE LL\_TIM\_EnableARRPreload

**LL\_TIM\_DisableARRPreload**

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableARRPreload(     TIM_TypeDef * TIMx)</code>
Function Description	Disable auto-reload (ARR) preload.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ARPE LL_TIM_DisableARRPreload</li> </ul>

**LL\_TIM\_IsEnabledARRPreload**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload(     TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether auto-reload (ARR) preload is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 ARPE LL_TIM_IsEnabledARRPreload</li> </ul>

**LL\_TIM\_SetClockDivision**

Function Name	<code>__STATIC_INLINE void LL_TIM_SetClockDivision(     TIM_TypeDef * TIMx, uint32_t ClockDivision)</code>
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	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>ClockDivision:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_TIM_CLOCKDIVISION_DIV1</li> <li>– LL_TIM_CLOCKDIVISION_DIV2</li> <li>– LL_TIM_CLOCKDIVISION_DIV4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 CKD LL_TIM_SetClockDivision</li> </ul>

**LL\_TIM\_GetClockDivision**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_GetClockDivision(     TIM_TypeDef * TIMx)</code>
---------------	--

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	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_CLOCKDIVISION_DIV1</li> <li>- LL_TIM_CLOCKDIVISION_DIV2</li> <li>- LL_TIM_CLOCKDIVISION_DIV4</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 CKD LL_TIM_GetClockDivision</li> </ul>

### LL\_TIM\_SetCounter

Function Name	<code>__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * *TIMx, uint32_t Counter)</code>
Function Description	Set the counter value.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>Counter:</b> Counter value (between Min_Data=0 and Max_Data=0xFFFF)</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CNT CNT LL_TIM_SetCounter</li> </ul>

### LL\_TIM\_GetCounter

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_GetCounter (TIM_TypeDef * TIMx)</code>
Function Description	Get the counter value.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Counter:</b> value (between Min_Data=0 and Max_Data=0xFFFF)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CNT CNT LL_TIM_GetCounter</li> </ul>

### LL\_TIM\_GetDirection

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_GetDirection (TIM_TypeDef * TIMx)</code>
Function Description	Get the current direction of the counter.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_TIM_COUNTERDIRECTION_UP</li> <li>– LL_TIM_COUNTERDIRECTION_DOWN</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DIR LL_TIM_GetDirection</li> </ul>

**LL\_TIM\_SetPrescaler**

Function Name	<code>__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)</code>
Function Description	Set the prescaler value.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Prescaler:</b> between Min_Data=0 and Max_Data=65535</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).</li> <li>• 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.</li> <li>• Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PSC PSC LL_TIM_SetPrescaler</li> </ul>

**LL\_TIM\_GetPrescaler**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (TIM_TypeDef * TIMx)</code>
Function Description	Get the prescaler value.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Prescaler:</b> value between Min_Data=0 and Max_Data=65535</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• PSC PSC LL_TIM_GetPrescaler</li> </ul>

**LL\_TIM\_SetAutoReload**

Function Name	<code>__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)</code>
Function Description	Set the auto-reload value.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>AutoReload:</b> between Min_Data=0 and Max_Data=65535</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

---

Notes	<ul style="list-style-type: none"> <li>The counter is blocked while the auto-reload value is null.</li> <li>Helper macro <code>__LL_TIM_CALC_ARR</code> can be used to calculate the AutoReload parameter</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li><code>ARR ARR LL_TIM_SetAutoReload</code></li> </ul>

### LL\_TIM\_GetAutoReload

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_GetAutoReload(     TIM_TypeDef * TIMx)</code>
Function Description	Get the auto-reload value.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Auto-reload:</b> value</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li><code>ARR ARR LL_TIM_GetAutoReload</code></li> </ul>

### LL\_TIM\_CC\_SetDMAReqTrigger

Function Name	<code>__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger(     TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)</code>
Function Description	Set the trigger of the capture/compare DMA request.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>DMAReqTrigger:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– <code>LL_TIM_CCDMAREQUEST_CC</code></li> <li>– <code>LL_TIM_CCDMAREQUEST_UPDATE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li><code>CR2 CCDS LL_TIM_CC_SetDMAReqTrigger</code></li> </ul>

### LL\_TIM\_CC\_GetDMAReqTrigger

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger(     TIM_TypeDef * TIMx)</code>
Function Description	Get actual trigger of the capture/compare DMA request.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– <code>LL_TIM_CCDMAREQUEST_CC</code></li> <li>– <code>LL_TIM_CCDMAREQUEST_UPDATE</code></li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li><code>CR2 CCDS LL_TIM_CC_GetDMAReqTrigger</code></li> </ul>

**LL\_TIM\_CC\_EnableChannel**

Function Name	<code>__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)</code>
Function Description	Enable capture/compare channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channels:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCER CC1E LL_TIM_CC_EnableChannel</li> <li>• CCER CC2E LL_TIM_CC_EnableChannel</li> <li>• CCER CC3E LL_TIM_CC_EnableChannel</li> <li>• CCER CC4E LL_TIM_CC_EnableChannel</li> </ul>

**LL\_TIM\_CC\_DisableChannel**

Function Name	<code>__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)</code>
Function Description	Disable capture/compare channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channels:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCER CC1E LL_TIM_CC_DisableChannel</li> <li>• CCER CC2E LL_TIM_CC_DisableChannel</li> <li>• CCER CC3E LL_TIM_CC_DisableChannel</li> <li>• CCER CC4E LL_TIM_CC_DisableChannel</li> </ul>

**LL\_TIM\_CC\_IsEnabledChannel**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)</code>
Function Description	Indicate whether channel(s) is(are) enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channels:</b> This parameter can be a combination of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCER CC1E LL_TIM_CC_IsEnabledChannel</li> <li>CCER CC2E LL_TIM_CC_IsEnabledChannel</li> <li>CCER CC3E LL_TIM_CC_IsEnabledChannel</li> <li>CCER CC4E LL_TIM_CC_IsEnabledChannel</li> </ul>

### LL\_TIM\_OC\_ConfigOutput

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)</code>
Function Description	Configure an output channel.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_TIM_CHANNEL_CH1</li> <li>LL_TIM_CHANNEL_CH2</li> <li>LL_TIM_CHANNEL_CH3</li> <li>LL_TIM_CHANNEL_CH4</li> </ul> </li> <li><b>Configuration:</b> This parameter must be a combination of all the following values: <ul style="list-style-type: none"> <li>LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCMR1 CC1S LL_TIM_OC_ConfigOutput</li> <li>CCMR1 CC2S LL_TIM_OC_ConfigOutput</li> <li>CCMR2 CC3S LL_TIM_OC_ConfigOutput</li> <li>CCMR2 CC4S LL_TIM_OC_ConfigOutput</li> <li>CCER CC1P LL_TIM_OC_ConfigOutput</li> <li>CCER CC2P LL_TIM_OC_ConfigOutput</li> <li>CCER CC3P LL_TIM_OC_ConfigOutput</li> <li>CCER CC4P LL_TIM_OC_ConfigOutput</li> <li>•</li> </ul>

### LL\_TIM\_OC\_SetMode

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)</code>
Function Description	Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_TIM_CHANNEL_CH1</li> <li>LL_TIM_CHANNEL_CH2</li> <li>LL_TIM_CHANNEL_CH3</li> <li>LL_TIM_CHANNEL_CH4</li> </ul> </li> <li><b>Mode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_TIM_OCMODE_FROZEN</li> <li>LL_TIM_OCMODE_ACTIVE</li> <li>LL_TIM_OCMODE_INACTIVE</li> <li>LL_TIM_OCMODE_TOGGLE</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- LL_TIM_OCMODE_FORCED_INACTIVE</li> <li>- LL_TIM_OCMODE_FORCED_ACTIVE</li> <li>- LL_TIM_OCMODE_PWM1</li> <li>- LL_TIM_OCMODE_PWM2</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1M LL_TIM_OC_SetMode</li> <li>• CCMR1 OC2M LL_TIM_OC_SetMode</li> <li>• CCMR2 OC3M LL_TIM_OC_SetMode</li> <li>• CCMR2 OC4M LL_TIM_OC_SetMode</li> </ul>

### LL\_TIM\_OC\_GetMode

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_GetMode(     TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Get the output compare mode of an output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_OCMODE_FROZEN</li> <li>- LL_TIM_OCMODE_ACTIVE</li> <li>- LL_TIM_OCMODE_INACTIVE</li> <li>- LL_TIM_OCMODE_TOGGLE</li> <li>- LL_TIM_OCMODE_FORCED_INACTIVE</li> <li>- LL_TIM_OCMODE_FORCED_ACTIVE</li> <li>- LL_TIM_OCMODE_PWM1</li> <li>- LL_TIM_OCMODE_PWM2</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1M LL_TIM_OC_SetMode</li> <li>• CCMR1 OC2M LL_TIM_OC_SetMode</li> <li>• CCMR2 OC3M LL_TIM_OC_SetMode</li> <li>• CCMR2 OC4M LL_TIM_OC_SetMode</li> </ul>

### LL\_TIM\_OC\_SetPolarity

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_SetPolarity(     TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)</code>
Function Description	Set the polarity of an output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li>• <b>Polarity:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_OCPOLARITY_HIGH</li> </ul> </li> </ul>

- LL\_TIM\_OCPOLARITY\_LOW

Return values

- **None:**
- CCER CC1P LL\_TIM\_OC\_SetPolarity
- CCER CC2P LL\_TIM\_OC\_SetPolarity
- CCER CC3P 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\_CH2
  - LL\_TIM\_CHANNEL\_CH3
  - 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 CC2P LL\_TIM\_OC\_GetPolarity
- CCER CC3P LL\_TIM\_OC\_GetPolarity
- CCER CC4P LL\_TIM\_OC\_GetPolarity

### **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	<code>__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Disable fast mode for the output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1FE LL_TIM_OC_DisableFast</li> <li>• CCMR1 OC2FE LL_TIM_OC_DisableFast</li> <li>• CCMR2 OC3FE LL_TIM_OC_DisableFast</li> <li>• CCMR2 OC4FE LL_TIM_OC_DisableFast</li> </ul>

**LL\_TIM\_OC\_IsEnabledFast**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Indicates whether fast mode is enabled for the output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1FE LL_TIM_OC_IsEnabledFast</li> <li>• CCMR1 OC2FE LL_TIM_OC_IsEnabledFast</li> <li>• CCMR2 OC3FE LL_TIM_OC_IsEnabledFast</li> <li>• CCMR2 OC4FE LL_TIM_OC_IsEnabledFast</li> <li>• </li> </ul>

**LL\_TIM\_OC\_EnablePreload**

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Enable compare register (TIMx_CCRx) preload for the output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

- 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	<code>__STATIC_INLINE void LL_TIM_OC_DisablePreload(     TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Disable compare register (TIMx_CCRx) preload for the output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1PE LL_TIM_OC_DisablePreload</li> <li>• CCMR1 OC2PE LL_TIM_OC_DisablePreload</li> <li>• CCMR2 OC3PE LL_TIM_OC_DisablePreload</li> <li>• CCMR2 OC4PE LL_TIM_OC_DisablePreload</li> </ul>

### **LL\_TIM\_OC\_IsEnabledPreload**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload(     TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload</li> <li>• CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload</li> <li>• CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload</li> <li>• CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload</li> <li>• </li> </ul>

### **LL\_TIM\_OC\_EnableClear**

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_EnableClear(     TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Enable clearing the output channel on an external event.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Channel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_TIM_CHANNEL_CH1</li> <li>– LL_TIM_CHANNEL_CH2</li> <li>– LL_TIM_CHANNEL_CH3</li> <li>– LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function can only be used in Output compare and PWM modes. It does not work in Forced mode.</li> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1CE LL_TIM_OC_EnableClear</li> <li>• CCMR1 OC2CE LL_TIM_OC_EnableClear</li> <li>• CCMR2 OC3CE LL_TIM_OC_EnableClear</li> <li>• CCMR2 OC4CE LL_TIM_OC_EnableClear</li> </ul>

### LL\_TIM\_OC\_DisableClear

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_DisableClear(   TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Disable clearing the output channel on an external event.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_TIM_CHANNEL_CH1</li> <li>– LL_TIM_CHANNEL_CH2</li> <li>– LL_TIM_CHANNEL_CH3</li> <li>– LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 OC1CE LL_TIM_OC_DisableClear</li> <li>• CCMR1 OC2CE LL_TIM_OC_DisableClear</li> <li>• CCMR2 OC3CE LL_TIM_OC_DisableClear</li> <li>• CCMR2 OC4CE LL_TIM_OC_DisableClear</li> </ul>

### LL\_TIM\_OC\_IsEnabledClear

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear(   TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Indicates clearing the output channel on an external event is enabled for the output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_TIM_CHANNEL_CH1</li> <li>– LL_TIM_CHANNEL_CH2</li> <li>– LL_TIM_CHANNEL_CH3</li> <li>– LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function enables clearing the output channel on an external event.</li> <li>This function can only be used in Output compare and PWM modes. It does not work in Forced mode.</li> <li>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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCMR1 OC1CE LL_TIM_OC_IsEnabledClear</li> <li>CCMR1 OC2CE LL_TIM_OC_IsEnabledClear</li> <li>CCMR2 OC3CE LL_TIM_OC_IsEnabledClear</li> <li>CCMR2 OC4CE LL_TIM_OC_IsEnabledClear</li> <li>•</li> </ul>

### LL\_TIM\_OC\_SetCompareCH1

Function Name	<b><code>__STATIC_INLINE void LL_TIM_OC_SetCompareCH1( TIM_TypeDef * TIMx, uint32_t CompareValue)</code></b>
Function Description	Set compare value for output channel 1 (TIMx_CCR1).
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>CompareValue:</b> between Min_Data=0 and Max_Data=65535</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR1 CCR1 LL_TIM_OC_SetCompareCH1</li> </ul>

### LL\_TIM\_OC\_SetCompareCH2

Function Name	<b><code>__STATIC_INLINE void LL_TIM_OC_SetCompareCH2( TIM_TypeDef * TIMx, uint32_t CompareValue)</code></b>
Function Description	Set compare value for output channel 2 (TIMx_CCR2).
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>CompareValue:</b> between Min_Data=0 and Max_Data=65535</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR2 CCR2 LL_TIM_OC_SetCompareCH2</li> </ul>

**LL\_TIM\_OC\_SetCompareCH3**

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_SetCompareCH3(   TIM_TypeDef * TIMx, uint32_t CompareValue)</code>
Function Description	Set compare value for output channel 3 (TIMx_CCR3).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>CompareValue:</b> between Min_Data=0 and Max_Data=65535</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR3 CCR3 LL_TIM_OC_SetCompareCH3</li> </ul>

**LL\_TIM\_OC\_SetCompareCH4**

Function Name	<code>__STATIC_INLINE void LL_TIM_OC_SetCompareCH4(   TIM_TypeDef * TIMx, uint32_t CompareValue)</code>
Function Description	Set compare value for output channel 4 (TIMx_CCR4).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>CompareValue:</b> between Min_Data=0 and Max_Data=65535</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR4 CCR4 LL_TIM_OC_SetCompareCH4</li> </ul>

**LL\_TIM\_OC\_GetCompareCH1**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1(   TIM_TypeDef * TIMx)</code>
Function Description	Get compare value (TIMx_CCR1) set for output channel 1.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>CompareValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR1 CCR1 LL_TIM_OC_GetCompareCH1</li> </ul>

**LL\_TIM\_OC\_GetCompareCH2**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (TIM_TypeDef * TIMx)</code></b>
Function Description	Get compare value (TIMx_CCR2) set for output channel 2.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>CompareValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR2 CCR2 LL_TIM_OC_GetCompareCH2</li> </ul>

**LL\_TIM\_OC\_GetCompareCH3**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (TIM_TypeDef * TIMx)</code></b>
Function Description	Get compare value (TIMx_CCR3) set for output channel 3.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>CompareValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR3 CCR3 LL_TIM_OC_GetCompareCH3</li> </ul>

**LL\_TIM\_OC\_GetCompareCH4**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (TIM_TypeDef * TIMx)</code></b>
Function Description	Get compare value (TIMx_CCR4) set for output channel 4.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>CompareValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCR4 CCR4 LL_TIM_OC_GetCompareCH4</li> </ul>

**LL\_TIM\_IC\_Config**

Function Name	<code>__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)</code>
Function Description	Configure input channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li>• <b>Configuration:</b> This parameter must be a combination of all the following values: <ul style="list-style-type: none"> <li>- LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC</li> <li>- LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8</li> <li>- LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8</li> <li>- LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 CC1S LL_TIM_IC_Config</li> <li>• CCMR1 IC1PSC LL_TIM_IC_Config</li> <li>• CCMR1 IC1F LL_TIM_IC_Config</li> <li>• CCMR1 CC2S LL_TIM_IC_Config</li> <li>• CCMR1 IC2PSC LL_TIM_IC_Config</li> <li>• CCMR1 IC2F LL_TIM_IC_Config</li> <li>• CCMR2 CC3S LL_TIM_IC_Config</li> <li>• CCMR2 IC3PSC LL_TIM_IC_Config</li> <li>• CCMR2 IC3F LL_TIM_IC_Config</li> <li>• CCMR2 CC4S LL_TIM_IC_Config</li> <li>• CCMR2 IC4PSC LL_TIM_IC_Config</li> <li>• CCMR2 IC4F LL_TIM_IC_Config</li> <li>• CCER CC1P LL_TIM_IC_Config</li> <li>• CCER CC1NP LL_TIM_IC_Config</li> <li>• CCER CC2P LL_TIM_IC_Config</li> <li>• CCER CC2NP LL_TIM_IC_Config</li> <li>• CCER CC3P LL_TIM_IC_Config</li> <li>• CCER CC3NP LL_TIM_IC_Config</li> <li>• CCER CC4P LL_TIM_IC_Config</li> <li>• CCER CC4NP LL_TIM_IC_Config</li> </ul>

**LL\_TIM\_IC\_SetActiveInput**

Function Name	<code>__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)</code>
Function Description	Set the active input.

Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>Channel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li><b>ICActiveInput:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_ACTIVEINPUT_DIRECTTI</li> <li>- LL_TIM_ACTIVEINPUT_INDIRECTTI</li> <li>- LL_TIM_ACTIVEINPUT_TRC</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCMR1 CC1S LL_TIM_IC_SetActiveInput</li> <li>CCMR1 CC2S LL_TIM_IC_SetActiveInput</li> <li>CCMR2 CC3S LL_TIM_IC_SetActiveInput</li> <li>CCMR2 CC4S LL_TIM_IC_SetActiveInput</li> </ul>

### LL\_TIM\_IC\_GetActiveInput

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput(   TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Get the current active input.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>Channel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_ACTIVEINPUT_DIRECTTI</li> <li>- LL_TIM_ACTIVEINPUT_INDIRECTTI</li> <li>- LL_TIM_ACTIVEINPUT_TRC</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCMR1 CC1S LL_TIM_IC_GetActiveInput</li> <li>CCMR1 CC2S LL_TIM_IC_GetActiveInput</li> <li>CCMR2 CC3S LL_TIM_IC_GetActiveInput</li> <li>CCMR2 CC4S LL_TIM_IC_GetActiveInput</li> </ul>

### LL\_TIM\_IC\_SetPrescaler

Function Name	<code>__STATIC_INLINE void LL_TIM_IC_SetPrescaler(   TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)</code>
Function Description	Set the prescaler of input channel.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>Channel:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li><b>ICPrescaler:</b> This parameter can be one of the following</li> </ul>

	<p>values:</p> <ul style="list-style-type: none"> <li>- LL_TIM_ICPSC_DIV1</li> <li>- LL_TIM_ICPSC_DIV2</li> <li>- LL_TIM_ICPSC_DIV4</li> <li>- LL_TIM_ICPSC_DIV8</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 IC1PSC LL_TIM_IC_SetPrescaler</li> <li>• CCMR1 IC2PSC LL_TIM_IC_SetPrescaler</li> <li>• CCMR2 IC3PSC LL_TIM_IC_SetPrescaler</li> <li>• CCMR2 IC4PSC LL_TIM_IC_SetPrescaler</li> </ul>

### LL\_TIM\_IC\_GetPrescaler

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Get the current prescaler value acting on an input channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_ICPSC_DIV1</li> <li>- LL_TIM_ICPSC_DIV2</li> <li>- LL_TIM_ICPSC_DIV4</li> <li>- LL_TIM_ICPSC_DIV8</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCMR1 IC1PSC LL_TIM_IC_GetPrescaler</li> <li>• CCMR1 IC2PSC LL_TIM_IC_GetPrescaler</li> <li>• CCMR2 IC3PSC LL_TIM_IC_GetPrescaler</li> <li>• CCMR2 IC4PSC LL_TIM_IC_GetPrescaler</li> </ul>

### LL\_TIM\_IC\_SetFilter

Function Name	<code>__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)</code>
Function Description	Set the input filter duration.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li>• <b>ICFilter:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_IC_FILTER_FDIV1</li> <li>- LL_TIM_IC_FILTER_FDIV1_N2</li> <li>- LL_TIM_IC_FILTER_FDIV1_N4</li> <li>- LL_TIM_IC_FILTER_FDIV1_N8</li> <li>- LL_TIM_IC_FILTER_FDIV2_N6</li> </ul> </li> </ul>

- 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	<code>__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IC_Polarity)</code>
Function Description	Set the input channel polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li>• <b>IC_Polarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_IC_POLARITY_RISING</li> <li>- LL_TIM_IC_POLARITY_FALLING</li> <li>- LL_TIM_IC_POLARITY_BOTHEDGE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCER CC1P LL_TIM_IC_SetPolarity</li> <li>• CCER CC1NP LL_TIM_IC_SetPolarity</li> <li>• CCER CC2P LL_TIM_IC_SetPolarity</li> <li>• CCER CC2NP LL_TIM_IC_SetPolarity</li> <li>• CCER CC3P LL_TIM_IC_SetPolarity</li> <li>• CCER CC3NP LL_TIM_IC_SetPolarity</li> <li>• CCER CC4P LL_TIM_IC_SetPolarity</li> <li>• CCER CC4NP LL_TIM_IC_SetPolarity</li> </ul>

**LL\_TIM\_IC\_GetPolarity**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)</code>
Function Description	Get the current input channel polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_IC_POLARITY_RISING</li> <li>- LL_TIM_IC_POLARITY_FALLING</li> <li>- LL_TIM_IC_POLARITY_BOTHEDGE</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CCER CC1P LL_TIM_IC_GetPolarity</li> <li>• CCER CC1NP LL_TIM_IC_GetPolarity</li> <li>• CCER CC2P LL_TIM_IC_GetPolarity</li> <li>• CCER CC2NP LL_TIM_IC_GetPolarity</li> <li>• CCER CC3P LL_TIM_IC_GetPolarity</li> <li>• CCER CC3NP LL_TIM_IC_GetPolarity</li> <li>• CCER CC4P LL_TIM_IC_GetPolarity</li> <li>• CCER CC4NP LL_TIM_IC_GetPolarity</li> </ul>

**LL\_TIM\_IC\_EnableXORCombination**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)</code></b>
Function Description	Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TI1S LL_TIM_IC_EnableXORCombination</li> </ul>

**LL\_TIM\_IC\_DisableXORCombination**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)</code></b>
Function Description	Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TI1S LL_TIM_IC_DisableXORCombination</li> </ul>

**LL\_TIM\_IC\_IsEnabledXORCombination**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (TIM_TypeDef * TIMx)</code></b>
Function Description	Indicates whether the TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TI1S LL_TIM_IC_IsEnabledXORCombination</li> </ul>

**LL\_TIM\_IC\_GetCaptureCH1**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)</code></b>
Function Description	Get captured value for input channel 1.

Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>CapturedValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR1 CCR1 LL_TIM_IC_GetCaptureCH1</li> </ul>

### LL\_TIM\_IC\_GetCaptureCH2

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2(                   TIM_TypeDef * TIMx)</code>
Function Description	Get captured value for input channel 2.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>CapturedValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR2 CCR2 LL_TIM_IC_GetCaptureCH2</li> </ul>

### LL\_TIM\_IC\_GetCaptureCH3

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3(                   TIM_TypeDef * TIMx)</code>
Function Description	Get captured value for input channel 3.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>CapturedValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR3 CCR3 LL_TIM_IC_GetCaptureCH3</li> </ul>

### LL\_TIM\_IC\_GetCaptureCH4

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4(                   TIM_TypeDef * TIMx)</code>
Function Description	Get captured value for input channel 4.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>CapturedValue:</b> (between Min_Data=0 and Max_Data=65535)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CCR4 CCR4 LL_TIM_IC_GetCaptureCH4</li> </ul>

### LL\_TIM\_EnableExternalClock

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableExternalClock(     TIM_TypeDef * TIMx)</code>
Function Description	Enable external clock mode 2.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.</li> <li>Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR ECE LL_TIM_EnableExternalClock</li> </ul>

### LL\_TIM\_DisableExternalClock

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableExternalClock(     TIM_TypeDef * TIMx)</code>
Function Description	Disable external clock mode 2.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR ECE LL_TIM_DisableExternalClock</li> </ul>

### LL\_TIM\_IsEnabledExternalClock

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock(     TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether external clock mode 2 is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR ECE LL_TIM_IsEnabledExternalClock</li> </ul>

### LL\_TIM\_SetClockSource

Function Name	<b>__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)</b>
Function Description	Set the clock source of the counter clock.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>ClockSource:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CLOCKSOURCE_INTERNAL</li> <li>- LL_TIM_CLOCKSOURCE_EXT_MODE1</li> <li>- LL_TIM_CLOCKSOURCE_EXT_MODE2</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> <li>Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1.</li> <li>Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR SMS LL_TIM_SetClockSource</li> <li>SMCR ECE LL_TIM_SetClockSource</li> </ul>

### LL\_TIM\_SetEncoderMode

Function Name	<b>__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)</b>
Function Description	Set the encoder interface mode.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>EncoderMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_ENCODERMODE_X2_TI1</li> <li>- LL_TIM_ENCODERMODE_X2_TI2</li> <li>- LL_TIM_ENCODERMODE_X4_TI12</li> </ul> </li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR SMS LL_TIM_SetEncoderMode</li> </ul>

### LL\_TIM\_SetTriggerOutput

Function Name	<code>__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)</code>
Function Description	Set the trigger output (TRGO) used for timer synchronization .
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>TimerSynchronization:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_TRGO_RESET</li> <li>- LL_TIM_TRGO_ENABLE</li> <li>- LL_TIM_TRGO_UPDATE</li> <li>- LL_TIM_TRGO_CC1IF</li> <li>- LL_TIM_TRGO_OC1REF</li> <li>- LL_TIM_TRGO_OC2REF</li> <li>- LL_TIM_TRGO_OC3REF</li> <li>- LL_TIM_TRGO_OC4REF</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 MMS LL_TIM_SetTriggerOutput</li> </ul>

### LL\_TIM\_SetSlaveMode

Function Name	<code>__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)</code>
Function Description	Set the synchronization mode of a slave timer.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>SlaveMode:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_SLAVEMODE_DISABLED</li> <li>- LL_TIM_SLAVEMODE_RESET</li> <li>- LL_TIM_SLAVEMODE_GATED</li> <li>- LL_TIM_SLAVEMODE_TRIGGER</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.</li> </ul>

- Reference Manual to LL API cross reference:
- SMCR SMS LL\_TIM\_SetSlaveMode

### **LL\_TIM\_SetTriggerInput**

Function Name	<code>__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)</code>
Function Description	Set the selects the trigger input to be used to synchronize the counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> <li>• <b>TriggerInput:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_TS_ITR0</li> <li>- LL_TIM_TS_ITR1</li> <li>- LL_TIM_TS_ITR2</li> <li>- LL_TIM_TS_ITR3</li> <li>- LL_TIM_TS_TI1F_ED</li> <li>- LL_TIM_TS_TI1FP1</li> <li>- LL_TIM_TS_TI2FP2</li> <li>- LL_TIM_TS_ETRF</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SMCR TS LL_TIM_SetTriggerInput</li> </ul>

### **LL\_TIM\_EnableMasterSlaveMode**

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)</code>
Function Description	Enable the Master/Slave mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SMCR MSM LL_TIM_EnableMasterSlaveMode</li> </ul>

### **LL\_TIM\_DisableMasterSlaveMode**

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)</code>
Function Description	Disable the Master/Slave mode.

---

Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR MSM LL_TIM_DisableMasterSlaveMode</li> </ul>

### LL\_TIM\_IsEnabledMasterSlaveMode

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the Master/Slave mode is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>SMCR MSM LL_TIM_IsEnabledMasterSlaveMode</li> </ul>

### LL\_TIM\_ConfigETR

Function Name	<code>__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)</code>
Function Description	Configure the external trigger (ETR) input.
Parameters	<ul style="list-style-type: none"> <li><b>TIMx:</b> Timer instance</li> <li><b>ETRPolarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_ETR_POLARITY_NONINVERTED</li> <li>- LL_TIM_ETR_POLARITY_INVERTED</li> </ul> </li> <li><b>ETRPrescaler:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_ETR_PRESCALER_DIV1</li> <li>- LL_TIM_ETR_PRESCALER_DIV2</li> <li>- LL_TIM_ETR_PRESCALER_DIV4</li> <li>- LL_TIM_ETR_PRESCALER_DIV8</li> </ul> </li> <li><b>ETRFilter:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_ETR_FILTER_FDIV1</li> <li>- LL_TIM_ETR_FILTER_FDIV1_N2</li> <li>- LL_TIM_ETR_FILTER_FDIV1_N4</li> <li>- LL_TIM_ETR_FILTER_FDIV1_N8</li> <li>- LL_TIM_ETR_FILTER_FDIV2_N6</li> <li>- LL_TIM_ETR_FILTER_FDIV2_N8</li> <li>- LL_TIM_ETR_FILTER_FDIV4_N6</li> </ul> </li> </ul>

- 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\_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
  - LL\_TIM\_DMABURST\_BASEADDR\_CCMR3
  - LL\_TIM\_DMABURST\_BASEADDR\_CCR5
  - LL\_TIM\_DMABURST\_BASEADDR\_CCR6
  - LL\_TIM\_DMABURST\_BASEADDR\_OR1
  - LL\_TIM\_DMABURST\_BASEADDR\_OR2
  - LL\_TIM\_DMABURST\_BASEADDR\_OR3

- **DMA\_BurstLength:** 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\_SetRemap**

## Function Name

```
STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef *  
TIMx, uint32_t Remap)
```

## Function Description

Remap TIM inputs (input channel, internal/external triggers).

## Parameters

- **TIMx:** Timer instance
- **Remap:** Remap params depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of OR registers.

## Return values

- **None:**

## Notes

- Macro IS\_TIM\_REMAP\_INSTANCE(TIMx) can be used to check whether or not some timer inputs can be remapped.

Reference Manual to  
LL API cross  
reference:

- TIM2\_OR ETR\_RMP LL\_TIM\_SetRemap
- TIM2\_OR TI4\_RMP LL\_TIM\_SetRemap
- TIM21\_OR ETR\_RMP LL\_TIM\_SetRemap
- TIM21\_OR TI1\_RMP LL\_TIM\_SetRemap
- TIM21\_OR TI2\_RMP LL\_TIM\_SetRemap
- TIM22\_OR ETR\_RMP LL\_TIM\_SetRemap
- TIM22\_OR TI1\_RMP LL\_TIM\_SetRemap
- TIM3\_OR ETR\_RMP LL\_TIM\_SetRemap

- `TIM3_OR TI1_RMP LL_TIM_SetRemap`
- `TIM3_OR TI2_RMP LL_TIM_SetRemap`
- `TIM3_OR TI4_RMP LL_TIM_SetRemap`

### `LL_TIM_SetOCRefClearInputSource`

Function Name	<code>__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource(     TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)</code>
Function Description	Set the OCREF clear source.
Parameters	<ul style="list-style-type: none"> <li>• <b><code>TIMx</code>:</b> Timer instance</li> <li>• <b><code>OCRefClearInputSource</code>:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- <code>LL_TIM_OCREF_CLR_INT_NC</code></li> <li>- <code>LL_TIM_OCREF_CLR_INT_ETR</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT</li> <li>• This function can only be used in Output compare and PWM modes.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SMCR OCCS LL_TIM_SetOCRefClearInputSource</li> </ul>

### `LL_TIM_ClearFlag_UPDATE`

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE(     TIM_TypeDef * TIMx)</code>
Function Description	Clear the update interrupt flag (UIF).
Parameters	<ul style="list-style-type: none"> <li>• <b><code>TIMx</code>:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR UIF LL_TIM_ClearFlag_UPDATE</li> </ul>

### `LL_TIM_IsActiveFlag_UPDATE`

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE(     TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b><code>TIMx</code>:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR UIF LL_TIM_IsActiveFlag_UPDATE</li> </ul>

**LL\_TIM\_ClearFlag\_CC1**

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 1 interrupt flag (CC1F).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC1IF LL_TIM_ClearFlag_CC1</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC1**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC1IF LL_TIM_IsActiveFlag_CC1</li> </ul>

**LL\_TIM\_ClearFlag\_CC2**

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 2 interrupt flag (CC2F).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC2IF LL_TIM_ClearFlag_CC2</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC2**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC2IF LL_TIM_IsActiveFlag_CC2</li> </ul>

**LL\_TIM\_ClearFlag\_CC3**

Function Name	<code>_STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 3 interrupt flag (CC3F).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC3IF LL_TIM_ClearFlag_CC3</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC3**

Function Name	<code>_STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC3IF LL_TIM_IsActiveFlag_CC3</li> </ul>

**LL\_TIM\_ClearFlag\_CC4**

Function Name	<code>_STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 4 interrupt flag (CC4F).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC4IF LL_TIM_ClearFlag_CC4</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC4**

Function Name	<code>_STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC4IF LL_TIM_IsActiveFlag_CC4</li> </ul>

**LL\_TIM\_ClearFlag\_TRIG**

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)</code>
Function Description	Clear the trigger interrupt flag (TIF).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR TIF LL_TIM_ClearFlag_TRIG</li> </ul>

**LL\_TIM\_IsActiveFlag\_TRIG**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR TIF LL_TIM_IsActiveFlag_TRIG</li> </ul>

**LL\_TIM\_ClearFlag\_CC1OVR**

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC1OF LL_TIM_ClearFlag_CC1OVR</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC1OVR**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC1OF LL_TIM_IsActiveFlag_CC1OVR</li> </ul>

**LL\_TIM\_ClearFlag\_CC2OVR**

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC2OF LL_TIM_ClearFlag_CC2OVR</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC2OVR**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC2OF LL_TIM_IsActiveFlag_CC2OVR</li> </ul>

**LL\_TIM\_ClearFlag\_CC3OVR**

Function Name	<code>__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)</code>
Function Description	Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC3OF LL_TIM_ClearFlag_CC3OVR</li> </ul>

**LL\_TIM\_IsActiveFlag\_CC3OVR**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (TIM_TypeDef * TIMx)</code>
Function Description	Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• SR CC3OF LL_TIM_IsActiveFlag_CC3OVR</li> </ul>

reference:

### **LL\_TIM\_ClearFlag\_CC4OVR**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)</code></b>
Function Description	Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC4OF LL_TIM_ClearFlag_CC4OVR</li> </ul>

### **LL\_TIM\_IsActiveFlag\_CC4OVR**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (TIM_TypeDef * TIMx)</code></b>
Function Description	Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• SR CC4OF LL_TIM_IsActiveFlag_CC4OVR</li> </ul>

### **LL\_TIM\_EnableIT\_UPDATE**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)</code></b>
Function Description	Enable update interrupt (UIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER UIE LL_TIM_EnableIT_UPDATE</li> </ul>

### **LL\_TIM\_DisableIT\_UPDATE**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)</code></b>
Function Description	Disable update interrupt (UIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• DIER UIE LL_TIM_DisableIT_UPDATE</li> </ul>

reference:

### LL\_TIM\_IsEnabledIT\_UPDATE

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE(     TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the update interrupt (UIE) is enabled.
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER UIE LL_TIM_IsEnabledIT_UPDATE</li></ul>

### LL\_TIM\_EnableIT\_CC1

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef *     TIMx)</code>
Function Description	Enable capture/compare 1 interrupt (CC1IE).
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER CC1IE LL_TIM_EnableIT_CC1</li></ul>

### LL\_TIM\_DisableIT\_CC1

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef *     TIMx)</code>
Function Description	Disable capture/compare 1 interrupt (CC1IE).
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER CC1IE LL_TIM_DisableIT_CC1</li></ul>

### LL\_TIM\_IsEnabledIT\_CC1

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1(     TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER CC1IE LL_TIM_IsEnabledIT_CC1</li></ul>

reference:

### **LL\_TIM\_EnableIT\_CC2**

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)</code>
Function Description	Enable capture/compare 2 interrupt (CC2IE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC2IE LL_TIM_EnableIT_CC2</li> </ul>

### **LL\_TIM\_DisableIT\_CC2**

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * * TIMx)</code>
Function Description	Disable capture/compare 2 interrupt (CC2IE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC2IE LL_TIM_DisableIT_CC2</li> </ul>

### **LL\_TIM\_IsEnabledIT\_CC2**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC2IE LL_TIM_IsEnabledIT_CC2</li> </ul>

### **LL\_TIM\_EnableIT\_CC3**

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Enable capture/compare 3 interrupt (CC3IE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• DIER CC3IE LL_TIM_EnableIT_CC3</li> </ul>

reference:

### LL\_TIM\_DisableIT\_CC3

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Disable capture/compare 3 interrupt (CC3IE).
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER CC3IE LL_TIM_DisableIT_CC3</li></ul>

### LL\_TIM\_IsEnabledIT\_CC3

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER CC3IE LL_TIM_IsEnabledIT_CC3</li></ul>

### LL\_TIM\_EnableIT\_CC4

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Enable capture/compare 4 interrupt (CC4IE).
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>DIER CC4IE LL_TIM_EnableIT_CC4</li></ul>

### LL\_TIM\_DisableIT\_CC4

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Disable capture/compare 4 interrupt (CC4IE).
Parameters	<ul style="list-style-type: none"><li><b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"><li>DIER CC4IE LL_TIM_DisableIT_CC4</li></ul>

reference:

### **LL\_TIM\_IsEnabledIT\_CC4**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC4IE LL_TIM_IsEnabledIT_CC4</li> </ul>

### **LL\_TIM\_EnableIT\_TRIG**

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)</code>
Function Description	Enable trigger interrupt (TIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER TIE LL_TIM_EnableIT_TRIG</li> </ul>

### **LL\_TIM\_DisableIT\_TRIG**

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)</code>
Function Description	Disable trigger interrupt (TIE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER TIE LL_TIM_DisableIT_TRIG</li> </ul>

### **LL\_TIM\_IsEnabledIT\_TRIG**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the trigger interrupt (TIE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• DIER TIE LL_TIM_IsEnabledIT_TRIG</li> </ul>

reference:

### LL\_TIM\_EnableDMAReq\_UPDATE

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE(     TIM_TypeDef * TIMx)</code>
Function Description	Enable update DMA request (UDE).
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• DIER UDE LL_TIM_EnableDMAReq_UPDATE</li></ul>

### LL\_TIM\_DisableDMAReq\_UPDATE

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE(     TIM_TypeDef * TIMx)</code>
Function Description	Disable update DMA request (UDE).
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• DIER UDE LL_TIM_DisableDMAReq_UPDATE</li></ul>

### LL\_TIM\_IsEnabledDMAReq\_UPDATE

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE(TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the update DMA request (UDE) is enabled.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE</li></ul>

### LL\_TIM\_EnableDMAReq\_CC1

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1(     TIM_TypeDef * TIMx)</code>
Function Description	Enable capture/compare 1 DMA request (CC1DE).
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• DIER CC1DE LL_TIM_EnableDMAReq_CC1</li></ul>

**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	<code>_STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2</li> </ul>

**LL\_TIM\_EnableDMAReq\_CC3**

Function Name	<code>_STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Enable capture/compare 3 DMA request (CC3DE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC3DE LL_TIM_EnableDMAReq_CC3</li> </ul>

**LL\_TIM\_DisableDMAReq\_CC3**

Function Name	<code>_STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Disable capture/compare 3 DMA request (CC3DE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC3DE LL_TIM_DisableDMAReq_CC3</li> </ul>

**LL\_TIM\_IsEnabledDMAReq\_CC3**

Function Name	<code>_STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3</li> </ul>

**LL\_TIM\_EnableDMAReq\_CC4**

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Enable capture/compare 4 DMA request (CC4DE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC4DE LL_TIM_EnableDMAReq_CC4</li> </ul>

**LL\_TIM\_DisableDMAReq\_CC4**

Function Name	<code>__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Disable capture/compare 4 DMA request (CC4DE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC4DE LL_TIM_DisableDMAReq_CC4</li> </ul>

**LL\_TIM\_IsEnabledDMAReq\_CC4**

Function Name	<code>__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (TIM_TypeDef * TIMx)</code>
Function Description	Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4</li> </ul>

**LL\_TIM\_EnableDMAReq\_TRIG**

Function Name	<code>__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)</code>
Function Description	Enable trigger interrupt (TDE).
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• DIER TDE LL_TIM_EnableDMAReq_TRIG</li> </ul>

**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	<b><code>_STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)</code></b>
Function Description	Generate Capture/Compare 2 event.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• EGR CC2G LL_TIM_GenerateEvent_CC2</li></ul>

**LL\_TIM\_GenerateEvent\_CC3**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)</code></b>
Function Description	Generate Capture/Compare 3 event.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• EGR CC3G LL_TIM_GenerateEvent_CC3</li></ul>

**LL\_TIM\_GenerateEvent\_CC4**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)</code></b>
Function Description	Generate Capture/Compare 4 event.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• EGR CC4G LL_TIM_GenerateEvent_CC4</li></ul>

**LL\_TIM\_GenerateEvent\_TRIG**

Function Name	<b><code>_STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)</code></b>
Function Description	Generate trigger event.
Parameters	<ul style="list-style-type: none"><li>• <b>TIMx:</b> Timer instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• EGR TG LL_TIM_GenerateEvent_TRIG</li></ul>

**LL\_TIM\_DeInit**

Function Name	<b>ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)</b>
Function Description	Set TIMx registers to their reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: TIMx registers are de-initialized</li> <li>– ERROR: invalid TIMx instance</li> </ul> </li> </ul>

**LL\_TIM\_StructInit**

Function Name	<b>void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)</b>
Function Description	Set the fields of the time base unit configuration data structure to their default values.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIM_InitStruct:</b> pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_TIM\_Init**

Function Name	<b>ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, LL_TIM_InitTypeDef * TIM_InitStruct)</b>
Function Description	Configure the TIMx time base unit.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer Instance</li> <li>• <b>TIM_InitStruct:</b> pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: TIMx registers are de-initialized</li> <li>– ERROR: not applicable</li> </ul> </li> </ul>

**LL\_TIM\_OC\_StructInit**

Function Name	<b>void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)</b>
Function Description	Set the fields of the TIMx output channel configuration data structure to their default values.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIM_OC_InitStruct:</b> pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_TIM\_OC\_Init**

Function Name	<b>ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)</b>
Function Description	Configure the TIMx output channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>TIM_OC_InitStruct:</b> pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: TIMx output channel is initialized</li> <li>- ERROR: TIMx output channel is not initialized</li> </ul> </li> </ul>

### LL\_TIM\_IC\_StructInit

Function Name	<b>void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)</b>
Function Description	Set the fields of the TIMx input channel configuration data structure to their default values.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIM_ICInitStruct:</b> pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

### LL\_TIM\_IC\_Init

Function Name	<b>ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)</b>
Function Description	Configure the TIMx input channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer Instance</li> <li>• <b>Channel:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_TIM_CHANNEL_CH1</li> <li>- LL_TIM_CHANNEL_CH2</li> <li>- LL_TIM_CHANNEL_CH3</li> <li>- LL_TIM_CHANNEL_CH4</li> </ul> </li> <li>• <b>TIM_IC_InitStruct:</b> pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>- SUCCESS: TIMx output channel is initialized</li> <li>- ERROR: TIMx output channel is not initialized</li> </ul> </li> </ul>

### LL\_TIM\_ENCODER\_StructInit

Function Name	<b>void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)</b>
Function Description	Fills each TIM_EncoderInitStruct field with its default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIM_EncoderInitStruct:</b> pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_TIM\_ENCODER\_Init**

Function Name	<b>ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)</b>
Function Description	Configure the encoder interface of the timer instance.
Parameters	<ul style="list-style-type: none"> <li>• <b>TIMx:</b> Timer Instance</li> <li>• <b>TIM_EncoderInitStruct:</b> pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: TIMx registers are de-initialized</li> <li>– ERROR: not applicable</li> </ul> </li> </ul>

**74.3 TIM Firmware driver defines****74.3.1 TIM*****Active Input Selection***

LL_TIM_ACTIVEINPUT_DIRECTTI	ICx is mapped on TIx
LL_TIM_ACTIVEINPUT_INDIRECTTI	ICx is mapped on Tly
LL_TIM_ACTIVEINPUT_TRC	ICx is mapped on TRC

***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

***Channel***

LL_TIM_CHANNEL_CH1	Timer input/output channel 1
LL_TIM_CHANNEL_CH2	Timer input/output channel 2
LL_TIM_CHANNEL_CH3	Timer input/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 input
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
<b>Counter Mode</b>	
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

LL_TIM_DMABURST_BASEADDR_CCR1	address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR2	TIMx_CCR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR3	TIMx_CCR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR4	TIMx_CCR3 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
LL_TIM_DMABURST_BASEADDR_CCMR3	TIMx_CCMR3 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR5	TIMx_CCR5 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_CCR6	TIMx_CCR6 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR1	TIMx_OR1 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR2	TIMx_OR2 register is the DMA base address for DMA burst
LL_TIM_DMABURST_BASEADDR_OR3	TIMx_OR3 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 8 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 I

**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***

<code>LL_TIM_ETR_POLARITY_NONINVERTED</code>	ETR is non-inverted, active at high level or rising edge
<code>LL_TIM_ETR_POLARITY_INVERTED</code>	ETR is inverted, active at low level or falling edge

#### ***External Trigger Prescaler***

<code>LL_TIM_ETR_PRESCALER_DIV1</code>	ETR prescaler OFF
<code>LL_TIM_ETR_PRESCALER_DIV2</code>	ETR frequency is divided by 2
<code>LL_TIM_ETR_PRESCALER_DIV4</code>	ETR frequency is divided by 4
<code>LL_TIM_ETR_PRESCALER_DIV8</code>	ETR frequency is divided by 8

#### ***Get Flags Defines***

<code>LL_TIM_SR_UIF</code>	Update interrupt flag
<code>LL_TIM_SR_CC1IF</code>	Capture/compare 1 interrupt flag
<code>LL_TIM_SR_CC2IF</code>	Capture/compare 2 interrupt flag
<code>LL_TIM_SR_CC3IF</code>	Capture/compare 3 interrupt flag
<code>LL_TIM_SR_CC4IF</code>	Capture/compare 4 interrupt flag
<code>LL_TIM_SR_TIF</code>	Trigger interrupt flag
<code>LL_TIM_SR_CC1OF</code>	Capture/Compare 1 overcapture flag
<code>LL_TIM_SR_CC2OF</code>	Capture/Compare 2 overcapture flag
<code>LL_TIM_SR_CC3OF</code>	Capture/Compare 3 overcapture flag
<code>LL_TIM_SR_CC4OF</code>	Capture/Compare 4 overcapture flag

#### ***Input Configuration Prescaler***

<code>LL_TIM_ICPSC_DIV1</code>	No prescaler, capture is done each time an edge is detected on the capture input
<code>LL_TIM_ICPSC_DIV2</code>	Capture is done once every 2 events
<code>LL_TIM_ICPSC_DIV4</code>	Capture is done once every 4 events
<code>LL_TIM_ICPSC_DIV8</code>	Capture is done once every 8 events

#### ***Input Configuration Filter***

<code>LL_TIM_IC_FILTER_FDIV1</code>	No filter, sampling is done at fDTS
<code>LL_TIM_IC_FILTER_FDIV1_N2</code>	fSAMPLING=fCK_INT, N=2
<code>LL_TIM_IC_FILTER_FDIV1_N4</code>	fSAMPLING=fCK_INT, N=4
<code>LL_TIM_IC_FILTER_FDIV1_N8</code>	fSAMPLING=fCK_INT, N=8
<code>LL_TIM_IC_FILTER_FDIV2_N6</code>	fSAMPLING=fDTS/2, N=6
<code>LL_TIM_IC_FILTER_FDIV2_N8</code>	fSAMPLING=fDTS/2, N=8
<code>LL_TIM_IC_FILTER_FDIV4_N6</code>	fSAMPLING=fDTS/4, N=6
<code>LL_TIM_IC_FILTER_FDIV4_N8</code>	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
LL_TIM_IC_POLARITY_BOTHEDGE	The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not 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_TIE	Trigger interrupt enable

***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 OCx active high

LL\_TIM\_OCPOLARITY\_LOW OCx active low

***OCREF clear input selection***

LL\_TIM\_OCREF\_CLR\_INT\_NC OCREF\_CLR\_INT is not connected

LL\_TIM\_OCREF\_CLR\_INT\_ETR OCREF\_CLR\_INT is connected to ETRF

***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\_REPEATIVE Counter stops counting at the next update event

***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

***TIM21 External Trigger Remap***

LL\_TIM\_TIM21\_ETR\_RMP\_GPIO TIM21\_ETR is connected to Ored GPIO1

LL\_TIM\_TIM21\_ETR\_RMP\_COMP2 TIM21\_ETR is connected to COMP2\_OUT

LL\_TIM\_TIM21\_ETR\_RMP\_COMP1 TIM21\_ETR is connected to COMP1\_OUT

LL\_TIM\_TIM21\_ETR\_RMP\_LSE TIM21\_ETR is connected to LSE

***TIM21 External Input Ch1 Remap***

LL\_TIM\_TIM21\_TI1\_RMP\_GPIO TIM21\_TI1 is connected to Ored GPIO1

LL\_TIM\_TIM21\_TI1\_RMP\_RTC\_WK TIM21\_TI1 is connected to RTC\_WAKEUP

LL\_TIM\_TIM21\_TI1\_RMP\_HSE\_RTC TIM21\_TI1 is connected to HSE\_RTC

LL\_TIM\_TIM21\_TI1\_RMP\_MSI TIM21\_TI1 is connected to MSI

LL\_TIM\_TIM21\_TI1\_RMP\_LSE TIM21\_TI1 is connected to LSE

LL\_TIM\_TIM21\_TI1\_RMP\_LSI TIM21\_TI1 is connected to LSI

LL\_TIM\_TIM21\_TI1\_RMP\_COMP1 TIM21\_TI1 is connected to COMP1\_OUT

LL\_TIM\_TIM21\_TI1\_RMP\_MCO TIM21\_TI1 is connected to MCO

***TIM21 External Input Ch2 Remap***

LL\_TIM\_TIM21\_TI2\_RMP\_GPIO TIM21\_TI2 is connected to Ored GPIO1

LL\_TIM\_TIM21\_TI2\_RMP\_COMP2 TIM21\_TI2 is connected to COMP2\_OUT

***TIM22 External Trigger Remap***

LL_TIM_TIM22_ETR_RMP_GPIO	TIM22_ETR is connected to GPIO
LL_TIM_TIM22_ETR_RMP_COMP2	TIM22_ETR is connected to COMP2_OUT
LL_TIM_TIM22_ETR_RMP_COMP1	TIM22_ETR is connected to COMP1_OUT
LL_TIM_TIM22_ETR_RMP_LSE	TIM22_ETR is connected to LSE

***TIM22 External Input Ch1 Remap***

LL_TIM_TIM22_TI1_RMP_GPIO1	TIM22_TI1 is connected to GPIO1
LL_TIM_TIM22_TI1_RMP_COMP2	TIM22_TI1 is connected to COMP2_OUT
LL_TIM_TIM22_TI1_RMP_COMP1	TIM22_TI1 is connected to COMP1_OUT
LL_TIM_TIM22_TI1_RMP_GPIO2	TIM22_TI1 is connected to GPIO2

***TIM2 External Trigger Remap***

LL_TIM_TIM2_ETR_RMP_GPIO	TIM2_ETR is connected to Ored GPIO
LL_TIM_TIM2_ETR_RMP_HSI	TIM2_ETR is connected to HSI
LL_TIM_TIM2_ETR_RMP_HSI48	TIM2_ETR is connected to HSI48
LL_TIM_TIM2_ETR_RMP_LSE	TIM2_ETR is connected to LSE
LL_TIM_TIM2_ETR_RMP_COMP2	TIM2_ETR is connected to COMP2_OUT
LL_TIM_TIM2_ETR_RMP_COMP1	TIM2_ETR is connected to COMP1_OUT

***TIM2 Timer Input Ch4 Remap***

LL_TIM_TIM2_TI4_RMP_GPIO	TIM2 input capture 4 is connected to GPIO
LL_TIM_TIM2_TI4_RMP_COMP2	TIM2 input capture 4 is connected to COMP2_OUT
LL_TIM_TIM2_TI4_RMP_COMP1	TIM2 input capture 4 is connected to COMP1_OUT

***TIM3 External Trigger Remap***

LL_TIM_TIM3_ETR_RMP_GPIO	TIM3_ETR is connected to GPIO
LL_TIM_TIM3_ETR_RMP_HSI48DIV6	TIM3_ETR is connected to HSI48 divided by 6

***TIM3 External Inputs Remap***

LL_TIM_TIM3_TI_RMP_TI1_USB_SOF	TIM3_TI1 input is connected to USB_SOF
LL_TIM_TIM3_TI_RMP_TI1_GPIO	TIM3_TI1 input is connected to PE3, PA6, PC6 or PB4
LL_TIM_TIM3_TI_RMP_TI2_GPIO_DEF	Mapping PB5 to TIM22_CH2
LL_TIM_TIM3_TI_RMP_TI2_GPIOB5_AF4	Mapping PB5 to TIM3_CH2
LL_TIM_TIM3_TI_RMP_TI4_GPIO_DEF	Mapping PC9 to USB_OE
LL_TIM_TIM3_TI_RMP_TI4_GPIOC9_AF2	Mapping PC9 to TIM3_CH4

***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

<code>LL_TIM_TRGO_OC1REF</code>	OC1REF signal is used as trigger output
<code>LL_TIM_TRGO_OC2REF</code>	OC2REF signal is used as trigger output
<code>LL_TIM_TRGO_OC3REF</code>	OC3REF signal is used as trigger output
<code>LL_TIM_TRGO_OC4REF</code>	OC4REF signal is used as trigger output

#### ***Trigger Selection***

<code>LL_TIM_TS_ITR0</code>	Internal Trigger 0 (ITR0) is used as trigger input
<code>LL_TIM_TS_ITR1</code>	Internal Trigger 1 (ITR1) is used as trigger input
<code>LL_TIM_TS_ITR2</code>	Internal Trigger 2 (ITR2) is used as trigger input
<code>LL_TIM_TS_ITR3</code>	Internal Trigger 3 (ITR3) is used as trigger input
<code>LL_TIM_TS_TI1F_ED</code>	TI1 Edge Detector (TI1F_ED) is used as trigger input
<code>LL_TIM_TS_TI1FP1</code>	Filtered Timer Input 1 (TI1FP1) is used as trigger input
<code>LL_TIM_TS_TI2FP2</code>	Filtered Timer Input 2 (TI12P2) is used as trigger input
<code>LL_TIM_TS_ETRF</code>	Filtered external Trigger (ETRF) is used as trigger input

#### ***Update Source***

<code>LL_TIM_UPDATESOURCE_REGULAR</code>	Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request
<code>LL_TIM_UPDATESOURCE_COUNTER</code>	Only counter overflow/underflow generates an update request

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

<code>_LL_TIM_CALC_PSC</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• HELPER macro calculating the prescaler value to achieve the required counter clock frequency.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_TIMCLK_</code>: timer input clock frequency (in Hz)</li> <li>• <code>_CNTCLK_</code>: counter clock frequency (in Hz)</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• Prescaler: value (between Min_Data=0 and Max_Data=65535)</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• ex: <code>_LL_TIM_CALC_PSC (80000000, 1000000);</code></li> </ul>
<code>_LL_TIM_CALC_ARR</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• HELPER macro calculating the auto-reload value to achieve the required output signal frequency.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_TIMCLK_</code>: timer input clock frequency (in Hz)</li> <li>• <code>_PSC_</code>: prescaler</li> <li>• <code>_FREQ_</code>: output signal frequency (in Hz)</li> </ul> <p><b>Return value:</b></p>

- 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

**Bit Position Value**

TIM\_POSITION\_ICPSC field position in half register TIMx\_CCMRx (8 bits)

## 75 LL USART Generic Driver

### 75.1 USART Firmware driver registers structures

#### 75.1.1 LL\_USART\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t DataWidth*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t TransferDirection*
- *uint32\_t HardwareFlowControl*
- *uint32\_t OverSampling*

##### 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()**.
- ***uint32\_t LL\_USART\_InitTypeDef::OverSampling***  
Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of **USART\_LL\_EC\_OVERSAMPLING**. This feature can be modified afterwards using unitary function **LL\_USART\_SetOverSampling()**.

#### 75.1.2 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.

## 75.2 USART Firmware driver API description

### 75.2.1 Detailed description of functions

**LL\_USART\_Enable**

Function Name	<code>__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)</code>
Function Description	USART Enable.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UE LL_USART_Enable</li> </ul>

**LL\_USART\_Disable**

Function Name	<code>__STATIC_INLINE void LL_USART_Disable (USART_TypeDef</code>
---------------	---

**\* USARTx)**

Function Description	<b>USART Disable</b> (all USART prescalers and outputs are disabled)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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_ISR are set to their default values.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UE LL_USART_Disable</li> </ul>

**LL\_USART\_IsEnabled**

Function Name	<b>STATIC_INLINE uint32_t LL_USART_IsEnabled (USART_TypeDef * USARTx)</b>
Function Description	Indicate if USART is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UE LL_USART_IsEnabled</li> </ul>

**LL\_USART\_EnableInStopMode**

Function Name	<b>STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)</b>
Function Description	USART enabled in STOP Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC.</li> <li>• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 UESM LL_USART_EnableInStopMode</li> </ul>

**LL\_USART\_DisableInStopMode**

Function Name	<b>STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)</b>
Function Description	USART disabled in STOP Mode.

Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>When this function is disabled, USART is not able to wake up the MCU from Stop mode</li> <li>Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 UESM LL_USART_DisableInStopMode</li> </ul>

### LL\_USART\_IsEnabledInStopMode

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode(USART_TypeDef * USARTx)</code>
Function Description	Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 UESM LL_USART_IsEnabledInStopMode</li> </ul>

### LL\_USART\_EnableDirectionRx

Function Name	<code>__STATIC_INLINE void LL_USART_EnableDirectionRx(USART_TypeDef * USARTx)</code>
Function Description	Receiver Enable (Receiver is enabled and begins searching for a start bit)
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR1 RE LL_USART_EnableDirectionRx</li> </ul>

### LL\_USART\_DisableDirectionRx

Function Name	<code>__STATIC_INLINE void LL_USART_DisableDirectionRx(USART_TypeDef * USARTx)</code>
Function Description	Receiver Disable.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 RE LL_USART_DisableDirectionRx</li> </ul>

### LL\_USART\_EnableDirectionTx

Function Name	<code>__STATIC_INLINE void LL_USART_EnableDirectionTx(     USART_TypeDef * USARTx)</code>
Function Description	Transmitter Enable.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TE LL_USART_EnableDirectionTx</li> </ul>

### LL\_USART\_DisableDirectionTx

Function Name	<code>__STATIC_INLINE void LL_USART_DisableDirectionTx(     USART_TypeDef * USARTx)</code>
Function Description	Transmitter Disable.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 TE LL_USART_DisableDirectionTx</li> </ul>

### LL\_USART\_SetTransferDirection

Function Name	<code>__STATIC_INLINE void LL_USART_SetTransferDirection(     USART_TypeDef * USARTx, uint32_t TransferDirection)</code>
Function Description	Configure simultaneously enabled/disabled states of Transmitter and Receiver.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>TransferDirection:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_DIRECTION_NONE</li> <li>– LL_USART_DIRECTION_RX</li> <li>– LL_USART_DIRECTION_TX</li> <li>– LL_USART_DIRECTION_TX_RX</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 RE LL_USART_SetTransferDirection</li> <li>• CR1 TE LL_USART_SetTransferDirection</li> </ul>

**LL\_USART\_GetTransferDirection**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetTransferDirection(     USART_TypeDef * USARTx)</code>
Function Description	Return enabled/disabled states of Transmitter and Receiver.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_USART_DIRECTION_NONE</li> <li>– LL_USART_DIRECTION_RX</li> <li>– LL_USART_DIRECTION_TX</li> <li>– LL_USART_DIRECTION_TX_RX</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 RE LL_USART_GetTransferDirection</li> <li>• CR1 TE LL_USART_GetTransferDirection</li> </ul>

**LL\_USART\_SetParity**

Function Name	<code>__STATIC_INLINE void LL_USART_SetParity(     USART_TypeDef * USARTx, uint32_t Parity)</code>
Function Description	Configure Parity (enabled/disabled and parity mode if enabled).
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Parity:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_USART_PARITY_NONE</li> <li>– LL_USART_PARITY_EVEN</li> <li>– LL_USART_PARITY_ODD</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PS LL_USART_SetParity</li> <li>• CR1 PCE LL_USART_SetParity</li> </ul>

**LL\_USART\_GetParity**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetParity(     USART_TypeDef * USARTx)</code>
Function Description	Return Parity configuration (enabled/disabled and parity mode if enabled)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_USART_PARITY_NONE</li> <li>– LL_USART_PARITY_EVEN</li> <li>– LL_USART_PARITY_ODD</li> </ul> </li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR1 PS LL_USART_GetParity</li> </ul>

- 
- reference:
- CR1 PCE LL\_USART\_GetParity

### **LL\_USART\_SetWakeUpMethod**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetWakeUpMethod( USART_TypeDef * USARTx, uint32_t Method)</code></b>
Function Description	Set Receiver Wake Up method from Mute mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Method:</b> This parameter can be one of the following values:               <ul style="list-style-type: none"> <li>– LL_USART_WAKEUP_IDLELINE</li> <li>– LL_USART_WAKEUP_ADDRESSMARK</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 WAKE LL_USART_SetWakeUpMethod</li> </ul>

### **LL\_USART\_GetWakeUpMethod**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod( USART_TypeDef * USARTx)</code></b>
Function Description	Return Receiver Wake Up method from Mute mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:               <ul style="list-style-type: none"> <li>– LL_USART_WAKEUP_IDLELINE</li> <li>– LL_USART_WAKEUP_ADDRESSMARK</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 WAKE LL_USART_GetWakeUpMethod</li> </ul>

### **LL\_USART\_SetDataWidth**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetDataWidth( USART_TypeDef * USARTx, uint32_t DataWidth)</code></b>
Function Description	Set Word length (i.e.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>DataWidth:</b> This parameter can be one of the following values:               <ul style="list-style-type: none"> <li>– LL_USART_DATAWIDTH_7B</li> <li>– LL_USART_DATAWIDTH_8B</li> <li>– LL_USART_DATAWIDTH_9B</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 M0 LL_USART_SetDataWidth</li> <li>• CR1 M1 LL_USART_SetDataWidth</li> </ul>

**LL\_USART\_GetDataWidth**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetDataWidth(     USART_TypeDef * USARTx)</code>
Function Description	Return Word length (i.e.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_USART_DATAWIDTH_7B</li> <li>- LL_USART_DATAWIDTH_8B</li> <li>- LL_USART_DATAWIDTH_9B</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 M0 LL_USART_GetDataWidth</li> <li>• CR1 M1 LL_USART_GetDataWidth</li> </ul>

**LL\_USART\_EnableMuteMode**

Function Name	<code>__STATIC_INLINE void LL_USART_EnableMuteMode(     USART_TypeDef * USARTx)</code>
Function Description	Allow switch between Mute Mode and Active mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 MME LL_USART_EnableMuteMode</li> </ul>

**LL\_USART\_DisableMuteMode**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableMuteMode(     USART_TypeDef * USARTx)</code>
Function Description	Prevent Mute Mode use.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 MME LL_USART_DisableMuteMode</li> </ul>

**LL\_USART\_IsEnabledMuteMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode(     USART_TypeDef * USARTx)</code>
Function Description	Indicate if switch between Mute Mode and Active mode is allowed.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 MME LL_USART_IsEnabledMuteMode</li> </ul>

**LL\_USART\_SetOverSampling**

Function Name	<code>__STATIC_INLINE void LL_USART_SetOverSampling(     USART_TypeDef * USARTx, uint32_t OverSampling)</code>
Function Description	Set Oversampling to 8-bit or 16-bit mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>OverSampling:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_OVERSAMPLING_16</li> <li>– LL_USART_OVERSAMPLING_8</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 OVER8 LL_USART_SetOverSampling</li> </ul>

**LL\_USART\_GetOverSampling**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetOverSampling(     USART_TypeDef * USARTx)</code>
Function Description	Return Oversampling mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_OVERSAMPLING_16</li> <li>– LL_USART_OVERSAMPLING_8</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 OVER8 LL_USART_GetOverSampling</li> </ul>

**LL\_USART\_SetLastClkPulseOutput**

Function Name	<code>__STATIC_INLINE void LL_USART_SetLastClkPulseOutput(     USART_TypeDef * USARTx, uint32_t LastBitClockPulse)</code>
Function Description	Configure if Clock pulse of the last data bit is output to the SCLK pin or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>LastBitClockPulse:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_LASTCLKPULSE_NO_OUTPUT</li> <li>– LL_USART_LASTCLKPULSE_OUTPUT</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LBCL LL_USART_SetLastClkPulseOutput</li> </ul>

**LL\_USART\_GetLastClkPulseOutput**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (USART_TypeDef * USARTx)</code>
Function Description	Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_LASTCLKPULSE_NO_OUTPUT</li> <li>– LL_USART_LASTCLKPULSE_OUTPUT</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LBCL LL_USART_GetLastClkPulseOutput</li> </ul>

**LL\_USART\_SetClockPhase**

Function Name	<code>__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)</code>
Function Description	Select the phase of the clock output on the SCLK pin in synchronous mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>ClockPhase:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_PHASE_1EDGE</li> <li>– LL_USART_PHASE_2EDGE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 CPHA LL_USART_SetClockPhase</li> </ul>

**LL\_USART\_GetClockPhase**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetClockPhase (USART_TypeDef * USARTx)</code>
Function Description	Return phase of the clock output on the SCLK pin in synchronous mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_PHASE_1EDGE</li> <li>– LL_USART_PHASE_2EDGE</li> </ul> </li> </ul>

---

Notes	<ul style="list-style-type: none"> <li>Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 CPHA LL_USART_GetClockPhase</li> </ul>

### LL\_USART\_SetClockPolarity

Function Name	<code>__STATIC_INLINE void LL_USART_SetClockPolarity(     USART_TypeDef * USARTx, uint32_t ClockPolarity)</code>
Function Description	Select the polarity of the clock output on the SCLK pin in synchronous mode.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> <li><b>ClockPolarity:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_POLARITY_LOW</li> <li>– LL_USART_POLARITY_HIGH</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 CPOL LL_USART_SetClockPolarity</li> </ul>

### LL\_USART\_GetClockPolarity

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetClockPolarity(     USART_TypeDef * USARTx)</code>
Function Description	Return polarity of the clock output on the SCLK pin in synchronous mode.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_POLARITY_LOW</li> <li>– LL_USART_POLARITY_HIGH</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 CPOL LL_USART_GetClockPolarity</li> </ul>

### LL\_USART\_ConfigClock

Function Name	<code>__STATIC_INLINE void LL_USART_ConfigClock(     USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity,     uint32_t LBCPOutput)</code>
---------------	---

Function Description	Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> <li>• <b>Phase:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_PHASE_1EDGE</li> <li>- LL_USART_PHASE_2EDGE</li> </ul> </li> <li>• <b>Polarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_POLARITY_LOW</li> <li>- LL_USART_POLARITY_HIGH</li> </ul> </li> <li>• <b>LBCPOutput:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_LASTCLKPULSE_NO_OUTPUT</li> <li>- LL_USART_LASTCLKPULSE_OUTPUT</li> </ul> </li> </ul>
Return values	• <b>None:</b>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_INSTANCE(USARTTx) can be used to check whether or not Synchronous mode is supported by the USARTTx instance.</li> <li>• 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</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 CPHA LL_USART_ConfigClock</li> <li>• CR2 CPOL LL_USART_ConfigClock</li> <li>• CR2 LBCL LL_USART_ConfigClock</li> </ul>

### LL\_USART\_EnableSCLKOutput

Function Name	<b>STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)</b>
Function Description	Enable Clock output on SCLK pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	• <b>None:</b>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_INSTANCE(USARTTx) can be used to check whether or not Synchronous mode is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 CLKEN LL_USART_EnableSCLKOutput</li> </ul>

### LL\_USART\_DisableSCLKOutput

Function Name	<b>STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)</b>
Function Description	Disable Clock output on SCLK pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 CLKEN LL_USART_DisableSCLKOutput</li> </ul>

### LL\_USART\_IsEnabledSCLKOutput

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput(     USART_TypeDef * USARTx)</code>
Function Description	Indicate if Clock output on SCLK pin is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 CLKEN LL_USART_IsEnabledSCLKOutput</li> </ul>

### LL\_USART\_SetStopBitsLength

Function Name	<code>__STATIC_INLINE void LL_USART_SetStopBitsLength(     USART_TypeDef * USARTx, uint32_t StopBits)</code>
Function Description	Set the length of the stop bits.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> <li><b>StopBits:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_STOPBITS_0_5</li> <li>– LL_USART_STOPBITS_1</li> <li>– LL_USART_STOPBITS_1_5</li> <li>– LL_USART_STOPBITS_2</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR2 STOP LL_USART_SetStopBitsLength</li> </ul>

### LL\_USART\_GetStopBitsLength

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength(     USART_TypeDef * USARTx)</code>
Function Description	Retrieve the length of the stop bits.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_STOPBITS_0_5</li> <li>– LL_USART_STOPBITS_1</li> </ul> </li> </ul>

Reference Manual to  
LL API cross  
reference:

- LL\_USART\_STOPBITS\_1\_5
- LL\_USART\_STOPBITS\_2
- CR2 STOP LL\_USART\_GetStopBitsLength

### LL\_USART\_ConfigCharacter

Function Name	<code>__STATIC_INLINE void LL_USART_ConfigCharacter(     USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t     Parity, uint32_t StopBits)</code>
Function Description	Configure Character frame format (Datawidth, Parity control, Stop Bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>DataWidth:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_USART_DATAWIDTH_7B</li> <li>- LL_USART_DATAWIDTH_8B</li> <li>- LL_USART_DATAWIDTH_9B</li> </ul> </li> <li>• <b>Parity:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_USART_PARITY_NONE</li> <li>- LL_USART_PARITY_EVEN</li> <li>- LL_USART_PARITY_ODD</li> </ul> </li> <li>• <b>StopBits:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_USART_STOPBITS_0_5</li> <li>- LL_USART_STOPBITS_1</li> <li>- LL_USART_STOPBITS_1_5</li> <li>- LL_USART_STOPBITS_2</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 PS LL_USART_ConfigCharacter</li> <li>• CR1 PCE LL_USART_ConfigCharacter</li> <li>• CR1 M0 LL_USART_ConfigCharacter</li> <li>• CR1 M1 LL_USART_ConfigCharacter</li> <li>• CR2 STOP LL_USART_ConfigCharacter</li> </ul>

### LL\_USART\_SetTXRXSwap

Function Name	<code>__STATIC_INLINE void LL_USART_SetTXRXSwap(     USART_TypeDef * USARTx, uint32_t SwapConfig)</code>
Function Description	Configure TX/RX pins swapping setting.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>SwapConfig:</b> This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_USART_TXRX_STANDARD</li> </ul> </li> </ul>

– LL\_USART\_TXRX\_SWAPPED

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- CR2 SWAP LL\_USART\_SetTXRXSwap

### **LL\_USART\_GetTXRXSwap**

Function Name      **`__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap(  
                  USART_TypeDef * USARTx)`**

Function Description      Retrieve TX/RX pins swapping configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL\_USART\_TXRX\_STANDARD
  - LL\_USART\_TXRX\_SWAPPED

Reference Manual to  
LL API cross  
reference:

- CR2 SWAP LL\_USART\_GetTXRXSwap

### **LL\_USART\_SetRXPinLevel**

Function Name      **`__STATIC_INLINE void LL_USART_SetRXPinLevel(  
                  USART_TypeDef * USARTx, uint32_t PinInvMethod)`**

Function Description      Configure RX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL\_USART\_RXPIN\_LEVEL\_STANDARD
  - LL\_USART\_RXPIN\_LEVEL\_INVERTED

Return values

- **None:**

Reference Manual to  
LL API cross  
reference:

- CR2 RXINV LL\_USART\_SetRXPinLevel

### **LL\_USART\_GetRXPinLevel**

Function Name      **`__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel(  
                  USART_TypeDef * USARTx)`**

Function Description      Retrieve RX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL\_USART\_RXPIN\_LEVEL\_STANDARD
  - LL\_USART\_RXPIN\_LEVEL\_INVERTED

Reference Manual to  
LL API cross  
reference:

- CR2 RXINV LL\_USART\_GetRXPinLevel

**LL\_USART\_SetTXPinLevel**

Function Name	<code>__STATIC_INLINE void LL_USART_SetTXPinLevel(     USART_TypeDef * USARTx, uint32_t PinInvMethod)</code>
Function Description	Configure TX pin active level logic.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>PinInvMethod:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_TXPIN_LEVEL_STANDARD</li> <li>– LL_USART_TXPIN_LEVEL_INVERTED</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TXINV LL_USART_SetTXPinLevel</li> </ul>

**LL\_USART\_GetTXPinLevel**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel(     USART_TypeDef * USARTx)</code>
Function Description	Retrieve TX pin active level logic configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_TXPIN_LEVEL_STANDARD</li> <li>– LL_USART_TXPIN_LEVEL_INVERTED</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 TXINV LL_USART_GetTXPinLevel</li> </ul>

**LL\_USART\_SetBinaryDataLogic**

Function Name	<code>__STATIC_INLINE void LL_USART_SetBinaryDataLogic(     USART_TypeDef * USARTx, uint32_t DataLogic)</code>
Function Description	Configure Binary data logic.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>DataLogic:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_BINARY_LOGIC_POSITIVE</li> <li>– LL_USART_BINARY_LOGIC_NEGATIVE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 DATAINV LL_USART_SetBinaryDataLogic</li> </ul>

**LL\_USART\_GetBinaryDataLogic**

Function Name      **`_STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic(  
USART_TypeDef * USARTx)`**

Function Description      Retrieve Binary data configuration.

Parameters      • **USARTx:** USART Instance

Return values      • **Returned:** value can be one of the following values:  
   – LL\_USART\_BINARY\_LOGIC\_POSITIVE  
   – LL\_USART\_BINARY\_LOGIC\_NEGATIVE

Reference Manual to  
LL API cross  
reference:  
   • CR2 DATAINV LL\_USART\_GetBinaryDataLogic

**LL\_USART\_SetTransferBitOrder**

Function Name      **`_STATIC_INLINE void LL_USART_SetTransferBitOrder(  
USART_TypeDef * USARTx, uint32_t BitOrder)`**

Function Description      Configure transfer bit order (either Less or Most Significant Bit First)

Parameters      • **USARTx:** USART Instance

  • **BitOrder:** This parameter can be one of the following values:  
   – LL\_USART\_BITORDER\_LSBFIRST  
   – LL\_USART\_BITORDER\_MSBFIRST

Return values      • **None:**

Notes      • MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to  
LL API cross  
reference:  
   • CR2 MSBFIRST LL\_USART\_SetTransferBitOrder

**LL\_USART\_GetTransferBitOrder**

Function Name      **`_STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder(  
USART_TypeDef * USARTx)`**

Function Description      Return transfer bit order (either Less or Most Significant Bit First)

Parameters      • **USARTx:** USART Instance

Return values      • **Returned:** value can be one of the following values:  
   – LL\_USART\_BITORDER\_LSBFIRST  
   – LL\_USART\_BITORDER\_MSBFIRST

Notes      • MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to  
LL API cross  
reference:  
   • CR2 MSBFIRST LL\_USART\_GetTransferBitOrder

**LL\_USART\_EnableAutoBaudRate**

Function Name	<code>__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)</code>
Function Description	Enable Auto Baud-Rate Detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx)</code> can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ABREN <code>LL_USART_EnableAutoBaudRate</code></li> </ul>

**LL\_USART\_DisableAutoBaudRate**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)</code>
Function Description	Disable Auto Baud-Rate Detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx)</code> can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ABREN <code>LL_USART_DisableAutoBaudRate</code></li> </ul>

**LL\_USART\_IsEnabledAutoBaud**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (USART_TypeDef * USARTx)</code>
Function Description	Indicate if Auto Baud-Rate Detection mechanism is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx)</code> can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ABREN <code>LL_USART_IsEnabledAutoBaud</code></li> </ul>

**LL\_USART\_SetAutoBaudRateMode**

Function Name	<code>__STATIC_INLINE void LL_USART_SetAutoBaudRateMode(     USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)</code>
Function Description	Set Auto Baud-Rate mode bits.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>AutoBaudRateMode:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_USART_AUTOBAUD_DETECT_ON_STARTBIT</li> <li>- LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE</li> <li>- LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME</li> <li>- LL_USART_AUTOBAUD_DETECT_ON_55_FRAME</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USART x) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ABRMODE LL_USART_SetAutoBaudRateMode</li> </ul>

**LL\_USART\_GetAutoBaudRateMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode(     USART_TypeDef * USARTx)</code>
Function Description	Return Auto Baud-Rate mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_USART_AUTOBAUD_DETECT_ON_STARTBIT</li> <li>- LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE</li> <li>- LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME</li> <li>- LL_USART_AUTOBAUD_DETECT_ON_55_FRAME</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USART x) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ABRMODE LL_USART_GetAutoBaudRateMode</li> </ul>

**LL\_USART\_EnableRxTimeout**

Function Name	<code>__STATIC_INLINE void LL_USART_EnableRxTimeout(     USART_TypeDef * USARTx)</code>
---------------	---

Function Description	Enable Receiver Timeout.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RTOEN LL_USART_EnableRxTimeout</li> </ul>

### LL\_USART\_DisableRxTimeout

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableRxTimeout( USART_TypeDef * USARTx)</code></b>
Function Description	Disable Receiver Timeout.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RTOEN LL_USART_DisableRxTimeout</li> </ul>

### LL\_USART\_IsEnabledRxTimeout

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout( USART_TypeDef * USARTx)</code></b>
Function Description	Indicate if Receiver Timeout feature is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 RTOEN LL_USART_IsEnabledRxTimeout</li> </ul>

### LL\_USART\_ConfigNodeAddress

Function Name	<b><code>_STATIC_INLINE void LL_USART_ConfigNodeAddress( USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)</code></b>
Function Description	Set Address of the USART node.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> <li>• <b>AddressLen:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_USART_ADDRESS_DETECT_4B</li> <li>– LL_USART_ADDRESS_DETECT_7B</li> </ul> </li> <li>• <b>NodeAddress:</b> 4 or 7 bit Address of the USART node.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.</li> <li>• 4bits address node is used when 4-bit Address Detection is</li> </ul>

selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to  
LL API cross  
reference:

- CR2 ADD LL\_USART\_ConfigNodeAddress
- CR2 ADDM7 LL\_USART\_ConfigNodeAddress

### **LL\_USART\_GetNodeAddress**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_GetNodeAddress( USART_TypeDef * USARTx)</code></b>
Function Description	Return 8 bit Address of the USART node as set in ADD field of CR2.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Address:</b> of the USART node (Value between Min_Data=0 and Max_Data=255)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ADD LL_USART_GetNodeAddress</li> </ul>

### **LL\_USART\_GetNodeAddressLen**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen( USART_TypeDef * USARTx)</code></b>
Function Description	Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>- LL_USART_ADDRESS_DETECT_4B</li> <li>- LL_USART_ADDRESS_DETECT_7B</li> </ul> </li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 ADDM7 LL_USART_GetNodeAddressLen</li> </ul>

**LL\_USART\_EnableRTSHWFlowCtrl**

Function Name	<b><u>__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl(USART_TypeDef * USARTx)</u></b>
Function Description	Enable RTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 RTSE LL_USART_EnableRTSHWFlowCtrl</li> </ul>

**LL\_USART\_DisableRTSHWFlowCtrl**

Function Name	<b><u>__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl(USART_TypeDef * USARTx)</u></b>
Function Description	Disable RTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 RTSE LL_USART_DisableRTSHWFlowCtrl</li> </ul>

**LL\_USART\_EnableCTSHWFlowCtrl**

Function Name	<b><u>__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl(USART_TypeDef * USARTx)</u></b>
Function Description	Enable CTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSE LL_USART_EnableCTSHWFlowCtrl</li> </ul>

**LL\_USART\_DisableCTSHWFlowCtrl**

Function Name	<b><u>__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl(USART_TypeDef * USARTx)</u></b>
---------------	--

Function Description	Disable CTS HW Flow Control.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSE LL_USART_DisableCTSHWFlowCtrl</li> </ul>

### LL\_USART\_SetHWFlowCtrl

Function Name	<b>STATIC_INLINE void LL_USART_SetHWFlowCtrl( USART_TypeDef * USARTTx, uint32_t HardwareFlowControl)</b>
Function Description	Configure HW Flow Control mode (both CTS and RTS)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> <li>• <b>HardwareFlowControl:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_HWCONTROL_NONE</li> <li>- LL_USART_HWCONTROL_RTS</li> <li>- LL_USART_HWCONTROL_CTS</li> <li>- LL_USART_HWCONTROL_RTS_CTS</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 RTSE LL_USART_SetHWFlowCtrl</li> <li>• CR3 CTSE LL_USART_SetHWFlowCtrl</li> </ul>

### LL\_USART\_GetHWFlowCtrl

Function Name	<b>STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl( USART_TypeDef * USARTTx)</b>
Function Description	Return HW Flow Control configuration (both CTS and RTS)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_HWCONTROL_NONE</li> <li>- LL_USART_HWCONTROL_RTS</li> <li>- LL_USART_HWCONTROL_CTS</li> <li>- LL_USART_HWCONTROL_RTS_CTS</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR3 RTSE LL_USART_GetHWFlowCtrl</li> <li>• CR3 CTSE LL_USART_GetHWFlowCtrl</li> </ul>

reference:

### LL\_USART\_EnableOneBitSamp

Function Name	<b><code>__STATIC_INLINE void LL_USART_EnableOneBitSamp(         USART_TypeDef * USARTx)</code></b>
Function Description	Enable One bit sampling method.
Parameters	<ul style="list-style-type: none"><li><b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>CR3 ONEBIT LL_USART_EnableOneBitSamp</li></ul>

### LL\_USART\_DisableOneBitSamp

Function Name	<b><code>__STATIC_INLINE void LL_USART_DisableOneBitSamp(         USART_TypeDef * USARTx)</code></b>
Function Description	Disable One bit sampling method.
Parameters	<ul style="list-style-type: none"><li><b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>CR3 ONEBIT LL_USART_DisableOneBitSamp</li></ul>

### LL\_USART\_IsEnabledOneBitSamp

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp(         USART_TypeDef * USARTx)</code></b>
Function Description	Indicate if One bit sampling method is enabled.
Parameters	<ul style="list-style-type: none"><li><b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>CR3 ONEBIT LL_USART_IsEnabledOneBitSamp</li></ul>

### LL\_USART\_EnableOverrunDetect

Function Name	<b><code>__STATIC_INLINE void LL_USART_EnableOverrunDetect(         USART_TypeDef * USARTx)</code></b>
Function Description	Enable Overrun detection.
Parameters	<ul style="list-style-type: none"><li><b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li><b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>CR3 OVRDIS LL_USART_EnableOverrunDetect</li></ul>

**LL\_USART\_DisableOverrunDetect**

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableOverrunDetect(USART_TypeDef * USARTx)</code></b>
Function Description	Disable Overrun detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 OVRDIS LL_USART_DisableOverrunDetect</li> </ul>

**LL\_USART\_IsEnabledOverrunDetect**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect(USART_TypeDef * USARTx)</code></b>
Function Description	Indicate if Overrun detection is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 OVRDIS LL_USART_IsEnabledOverrunDetect</li> </ul>

**LL\_USART\_SetWKUPTYPE**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetWKUPTYPE(USART_TypeDef * USARTx, uint32_t Type)</code></b>
Function Description	Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Type:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>LL_USART_WAKEUP_ON_ADDRESS</code></li> <li>- <code>LL_USART_WAKEUP_ON_STARTBIT</code></li> <li>- <code>LL_USART_WAKEUP_ON_RXNE</code></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx)</code> can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 WUS LL_USART_SetWKUPTYPE</li> </ul>

**LL\_USART\_GetWKUPTYPE**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetWKUPTYPE(USART_TypeDef * USARTx)</code></b>
Function Description	Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values: <ul style="list-style-type: none"> <li>– LL_USART_WAKEUP_ON_ADDRESS</li> <li>– LL_USART_WAKEUP_ON_STARTBIT</li> <li>– LL_USART_WAKEUP_ON_RXNE</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_USART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 WUS LL_USART_GetWKUPType</li> </ul>

### LL\_USART\_SetBaudRate

Function Name	<code>__STATIC_INLINE void LL_USART_SetBaudRate(     USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t     OverSampling, uint32_t BaudRate)</code>
Function Description	Configure USART BRR register for achieving expected Baud Rate value.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> <li><b>PeriphClk:</b> Peripheral Clock</li> <li><b>OverSampling:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_USART_OVERSAMPLING_16</li> <li>– LL_USART_OVERSAMPLING_8</li> </ul> </li> <li><b>BaudRate:</b> Baud Rate</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values</li> <li>Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>BRR BRR LL_USART_SetBaudRate</li> </ul>

### LL\_USART\_GetBaudRate

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetBaudRate(     USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t     OverSampling)</code>
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	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> <li><b>PeriphClk:</b> Peripheral Clock</li> <li><b>OverSampling:</b> This parameter can be one of the following values:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- LL_USART_OVERSAMPLING_16</li> <li>- LL_USART_OVERSAMPLING_8</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Baud:</b> Rate</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• BRR BRR LL_USART_GetBaudRate</li> </ul>

### LL\_USART\_SetRxTimeout

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetRxTimeout( USART_TypeDef * USARTx, uint32_t Timeout)</code></b>
Function Description	Set Receiver Time Out Value (expressed in nb of bits duration)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Timeout:</b> Value between Min_Data=0x00 and Max_Data=0x00FFFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RTOR RTO LL_USART_SetRxTimeout</li> </ul>

### LL\_USART\_GetRxTimeout

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetRxTimeout( USART_TypeDef * USARTx)</code></b>
Function Description	Get Receiver Time Out Value (expressed in nb of bits duration)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x00FFFFFF</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RTOR RTO LL_USART_GetRxTimeout</li> </ul>

### LL\_USART\_SetBlockLength

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetBlockLength( USART_TypeDef * USARTx, uint32_t BlockLength)</code></b>
Function Description	Set Block Length value in reception.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>BlockLength:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RTOR BLEN LL_USART_SetBlockLength</li> </ul>

**LL\_USART\_GetBlockLength**

Function Name      **`__STATIC_INLINE uint32_t LL_USART_GetBlockLength  
(USART_TypeDef * USARTx)`**

Function Description      Get Block Length value in reception.

Parameters      • **USARTx:** USART Instance

Return values      • **Value:** between Min\_Data=0x00 and Max\_Data=0xFF

Reference Manual to  
LL API cross  
reference:  
• RTOR BLEN LL\_USART\_GetBlockLength

**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	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 IREN LL_USART_IsEnabledIrda</li> </ul>

### LL\_USART\_SetIrdaPowerMode

Function Name	<b><code>__STATIC_INLINE void LL_USART_SetIrdaPowerMode(     USART_TypeDef * USARTx, uint32_t PowerMode)</code></b>
Function Description	Configure IrDA Power Mode (Normal or Low Power)
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> <li><b>PowerMode:</b> This parameter can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_IRDA_POWER_NORMAL</li> <li>– LL_USART_IRDA_POWER_LOW</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 IRLP LL_USART_SetIrdaPowerMode</li> </ul>

### LL\_USART\_GetIrdaPowerMode

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode(     USART_TypeDef * USARTx)</code></b>
Function Description	Retrieve IrDA Power Mode configuration (Normal or Low Power)
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>– LL_USART_IRDA_POWER_NORMAL</li> <li>– LL_USART_PHASE_2EDGE</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 IRLP LL_USART_GetIrdaPowerMode</li> </ul>

### LL\_USART\_SetIrdaPrescaler

Function Name	<b><code>__STATIC_INLINE void LL_USART_SetIrdaPrescaler(     USART_TypeDef * USARTx, uint32_t PrescalerValue)</code></b>
Function Description	Set Irda prescaler value, used for dividing the USART clock source

	to achieve the Irda Low Power frequency (8 bits value)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> <li>• <b>PrescalerValue:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_IRDA_INSTANCE(USARTTx) can be used to check whether or not IrDA feature is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• GTPR PSC LL_USART_SetIrdaPrescaler</li> </ul>

### LL\_USART\_GetIrdaPrescaler

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler( USART_TypeDef * USARTx)</code></b>
Function Description	Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Irda:</b> prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_IRDA_INSTANCE(USARTTx) can be used to check whether or not IrDA feature is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• GTPR PSC LL_USART_GetIrdaPrescaler</li> </ul>

### LL\_USART\_EnableSmartcardNACK

Function Name	<b><code>_STATIC_INLINE void LL_USART_EnableSmartcardNACK( USART_TypeDef * USARTx)</code></b>
Function Description	Enable Smartcard NACK transmission.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTTx) can be used to check whether or not Smartcard feature is supported by the USARTTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 NACK LL_USART_EnableSmartcardNACK</li> </ul>

### LL\_USART\_DisableSmartcardNACK

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableSmartcardNACK( USART_TypeDef * USARTx)</code></b>
---------------	--

Function Description	Disable Smartcard NACK transmission.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 NACK LL_USART_DisableSmartcardNACK</li> </ul>

### LL\_USART\_IsEnabledSmartcardNACK

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (USART_TypeDef * USARTx)</code>
Function Description	Indicate if Smartcard NACK transmission is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 NACK LL_USART_IsEnabledSmartcardNACK</li> </ul>

### LL\_USART\_EnableSmartcard

Function Name	<code>__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)</code>
Function Description	Enable Smartcard mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 SCEN LL_USART_EnableSmartcard</li> </ul>

### LL\_USART\_DisableSmartcard

Function Name	<code>__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)</code>
Function Description	Disable Smartcard mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 SCEN LL_USART_DisableSmartcard</li> </ul>

### LL\_USART\_IsEnabledSmartcard

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (USART_TypeDef * USARTx)</code>
Function Description	Indicate if Smartcard mode is enabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 SCEN LL_USART_IsEnabledSmartcard</li> </ul>

### LL\_USART\_SetSmartcardAutoRetryCount

Function Name	<code>__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)</code>
Function Description	Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> <li><b>AutoRetryCount:</b> Value between Min_Data=0 and Max_Data=7</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> <li>This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number of erroneous reception trials, before generating a reception error (RXNE and PE bits set)</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount</li> </ul>

**LL\_USART\_GetSmartcardAutoRetryCount**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (USART_TypeDef * USARTx)</code>
Function Description	Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Smartcard:</b> Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount</li> </ul>

**LL\_USART\_SetSmartcardPrescaler**

Function Name	<code>__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)</code>
Function Description	Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>PrescalerValue:</b> Value between Min_Data=0 and Max_Data=31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• GTPR PSC LL_USART_SetSmartcardPrescaler</li> </ul>

**LL\_USART\_GetSmartcardPrescaler**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (USART_TypeDef * USARTx)</code>
Function Description	Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Smartcard:</b> prescaler value (Value between Min_Data=0 and Max_Data=31)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to	<ul style="list-style-type: none"> <li>• GTPR PSC LL_USART_GetSmartcardPrescaler</li> </ul>

LL API cross  
reference:

### **LL\_USART\_SetSmartcardGuardTime**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)</code></b>
Function Description	Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>GuardTime:</b> Value between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• GTPR GT LL_USART_SetSmartcardGuardTime</li> </ul>

### **LL\_USART\_GetSmartcardGuardTime**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (USART_TypeDef * USARTx)</code></b>
Function Description	Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Smartcard:</b> Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• GTPR GT LL_USART_GetSmartcardGuardTime</li> </ul>

### **LL\_USART\_EnableHalfDuplex**

Function Name	<b><code>_STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)</code></b>
Function Description	Enable Single Wire Half-Duplex mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.</li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CR3 HDSEL LL\_USART\_EnableHalfDuplex

### **LL\_USART\_DisableHalfDuplex**

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableHalfDuplex( USART_TypeDef * USARTx)</code></b>
Function Description	Disable Single Wire Half-Duplex mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 HDSEL LL_USART_DisableHalfDuplex</li> </ul>

### **LL\_USART\_IsEnabledHalfDuplex**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex( USART_TypeDef * USARTx)</code></b>
Function Description	Indicate if Single Wire Half-Duplex mode is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 HDSEL LL_USART_IsEnabledHalfDuplex</li> </ul>

### **LL\_USART\_SetLINBrkDetectionLen**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetLINBrkDetectionLen( USART_TypeDef * USARTx, uint32_t LINBDLength)</code></b>
Function Description	Set LIN Break Detection Length.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>LINBDLength:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– LL_USART_LINBREAK_DETECT_10B</li> <li>– LL_USART_LINBREAK_DETECT_11B</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CR2 LBDL LL\_USART\_SetLINBrkDetectionLen

### **LL\_USART\_GetLINBrkDetectionLen**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (USART_TypeDef * USARTx)</code>
Function Description	Return LIN Break Detection Length.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:           <ul style="list-style-type: none"> <li>– LL_USART_LINBREAK_DETECT_10B</li> <li>– LL_USART_LINBREAK_DETECT_11B</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LBDL LL_USART_GetLINBrkDetectionLen</li> </ul>

### **LL\_USART\_EnableLIN**

Function Name	<code>__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)</code>
Function Description	Enable LIN mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_EnableLIN</li> </ul>

### **LL\_USART\_DisableLIN**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)</code>
Function Description	Disable LIN mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_DisableLIN</li> </ul>

reference:

### **LL\_USART\_IsEnabledLIN**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledLIN(USART_TypeDef * USARTx)</code></b>
Function Description	Indicate if LIN mode is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_IsEnabledLIN</li> </ul>

### **LL\_USART\_SetDEDeassertionTime**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetDEDeassertionTime(USART_TypeDef * USARTx, uint32_t Time)</code></b>
Function Description	Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Time:</b> Value between Min_Data=0 and Max_Data=31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DEDT LL_USART_SetDEDeassertionTime</li> </ul>

### **LL\_USART\_GetDEDeassertionTime**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime(USART_TypeDef * USARTx)</code></b>
Function Description	Return DEDT (Driver Enable De-Assertion Time)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Time:</b> value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR1 DEDT LL_USART_GetDEDeassertionTime</li> </ul>

reference:

### **LL\_USART\_SetDEAssertionTime**

Function Name	<b><code>_STATIC_INLINE void LL_USART_SetDEAssertionTime(     USART_TypeDef * USARTx, uint32_t Time)</code></b>
Function Description	Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Time:</b> Value between Min_Data=0 and Max_Data=31</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DEAT LL_USART_SetDEAssertionTime</li> </ul>

### **LL\_USART\_GetDEAssertionTime**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime(     USART_TypeDef * USARTx)</code></b>
Function Description	Return DEAT (Driver Enable Assertion Time)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Time:</b> value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 DEAT LL_USART_GetDEAssertionTime</li> </ul>

### **LL\_USART\_EnableDEMode**

Function Name	<b><code>_STATIC_INLINE void LL_USART_EnableDEMode(     USART_TypeDef * USARTx)</code></b>
Function Description	Enable Driver Enable (DE) Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 DEM LL_USART_EnableDEMode</li> </ul>

**LL\_USART\_DisableDEMode**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)</code>
Function Description	Disable Driver Enable (DE) Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> <li>• CR3 DEM LL_USART_DisableDEMode</li> </ul>
Reference Manual to LL API cross reference:	

**LL\_USART\_IsEnabledDEMode**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (USART_TypeDef * USARTx)</code>
Function Description	Indicate if Driver Enable (DE) Mode is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> <li>• CR3 DEM LL_USART_IsEnabledDEMode</li> </ul>
Reference Manual to LL API cross reference:	

**LL\_USART\_SetDESignalPolarity**

Function Name	<code>__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)</code>
Function Description	Select Driver Enable Polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Polarity:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_DE_POLARITY_HIGH</li> <li>- LL_USART_DE_POLARITY_LOW</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> <li>• CR3 DEP LL_USART_SetDESignalPolarity</li> </ul>
Reference Manual to LL API cross reference:	

**LL\_USART\_GetDESignalPolarity**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity(     USART_TypeDef * USARTx)</code>
Function Description	Return Driver Enable Polarity.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Returned:</b> value can be one of the following values:             <ul style="list-style-type: none"> <li>- LL_USART_DE_POLARITY_HIGH</li> <li>- LL_USART_DE_POLARITY_LOW</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 DEP LL_USART_GetDESignalPolarity</li> </ul>

**LL\_USART\_ConfigAsyncMode**

Function Name	<code>__STATIC_INLINE void LL_USART_ConfigAsyncMode(     USART_TypeDef * USARTx)</code>
Function Description	Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• 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</li> <li>• Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_ConfigAsyncMode</li> <li>• CR2 CLKEN LL_USART_ConfigAsyncMode</li> <li>• CR3 SCEN LL_USART_ConfigAsyncMode</li> <li>• CR3 IREN LL_USART_ConfigAsyncMode</li> <li>• CR3 HDSEL LL_USART_ConfigAsyncMode</li> </ul>

**LL\_USART\_ConfigSyncMode**

Function Name	<code>__STATIC_INLINE void LL_USART_ConfigSyncMode</code>
---------------	---

**(USART\_TypeDef \* USARTx)**

Function Description	Perform basic configuration of USART for enabling use in Synchronous Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.</li> <li>• 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</li> <li>• Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_ConfigSyncMode</li> <li>• CR2 CLKEN LL_USART_ConfigSyncMode</li> <li>• CR3 SCEN LL_USART_ConfigSyncMode</li> <li>• CR3 IREN LL_USART_ConfigSyncMode</li> <li>• CR3 HDSEL LL_USART_ConfigSyncMode</li> </ul>

**LL\_USART\_ConfigLINMode**

Function Name	<b>STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)</b>
Function Description	Perform basic configuration of USART for enabling use in LIN Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> <li>• 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()</li> </ul>

Reference Manual to LL API cross reference:  <b>LL_USART_ConfigLINMode</b> <b>LL_USART_DisableHalfDuplex()</b> <b>LL_USART_DisableIrda()</b> <b>LL_USART_DisableSCLKOutput()</b> <b>LL_USART_DisableSmartcard()</b> <b>LL_USART_DisableLIN()</b> <b>LL_USART_EnableLIN()</b> <b>LL_USART_SetLINEN()</b> <b>LL_USART_SetSCEN()</b> <b>LL_USART_SetHDSEL()</b> <b>LL_USART_SetIREN()</b> <b>LL_USART_SetCLKEN()</b> <b>LL_USART_SetSTOP()</b> <b>LL_USART_SetLINEN()</b> <b>LL_USART_SetSCEN()</b> <b>LL_USART_SetHDSEL()</b> <b>LL_USART_SetIREN()</b> <b>LL_USART_SetCLKEN()</b> <b>LL_USART_SetSTOP()</b>	<ul style="list-style-type: none"> <li>functionClear IREN in CR3 using LL_USART_DisableIrda()</li> <li>functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex()</li> <li>functionSet LINEN in CR2 using LL_USART_EnableLIN() function</li> <li>• Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions</li> </ul> <ul style="list-style-type: none"> <li>• CR2 CLKEN LL_USART_ConfigLINMode</li> <li>• CR2 STOP LL_USART_ConfigLINMode</li> <li>• CR2 LINEN LL_USART_ConfigLINMode</li> <li>• CR3 IREN LL_USART_ConfigLINMode</li> <li>• CR3 SCEN LL_USART_ConfigLINMode</li> <li>• CR3 HDSEL LL_USART_ConfigLINMode</li> </ul>
--	--

### LL\_USART\_ConfigHalfDuplexMode

Function Name  <b>Function Name</b> <b>Function Description</b> <b>Parameters</b> <b>Return values</b> <b>Notes</b> <b>Reference Manual to LL API cross reference:</b>	<pre style="font-family: monospace; font-size: 10pt; margin: 0;"><b>STATIC_INLINE void LL_USART_ConfigHalfDuplexMode</b> <b>(USART_TypeDef * USARTx)</b></pre> <p>Perform basic configuration of USART for enabling use in Half Duplex Mode.</p> <ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>None:</b></li> <li>• 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.</li> <li>• Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.</li> <li>• 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</li> <li>• Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions</li> </ul> <ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_ConfigHalfDuplexMode</li> <li>• CR2 CLKEN LL_USART_ConfigHalfDuplexMode</li> <li>• CR3 HDSEL LL_USART_ConfigHalfDuplexMode</li> <li>• CR3 SCEN LL_USART_ConfigHalfDuplexMode</li> <li>• CR3 IREN LL_USART_ConfigHalfDuplexMode</li> </ul>
---	---

### LL\_USART\_ConfigSmartcardMode

Function Name  <b>Function Name</b>	<pre style="font-family: monospace; font-size: 10pt; margin: 0;"><b>STATIC_INLINE void LL_USART_ConfigSmartcardMode</b></pre>
---	---

**(USART\_TypeDef \* USARTx)**

Function Description	Perform basic configuration of USART for enabling use in Smartcard Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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).</li> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> <li>• 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</li> <li>• Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_ConfigSmartcardMode</li> <li>• CR2 STOP LL_USART_ConfigSmartcardMode</li> <li>• CR2 CLKEN LL_USART_ConfigSmartcardMode</li> <li>• CR3 HDSEL LL_USART_ConfigSmartcardMode</li> <li>• CR3 SCEN LL_USART_ConfigSmartcardMode</li> </ul>

**LL\_USART\_ConfigIrdaMode**

Function Name	<b>_STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)</b>
Function Description	Perform basic configuration of USART for enabling use in Irda Mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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).</li> <li>• Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.</li> <li>• Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using</li> </ul>

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	<b><code>__STATIC_INLINE void LL_USART_ConfigMultiProcessMode( USART_TypeDef * USARTx)</code></b>
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	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• 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</li> <li>• Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LINEN LL_USART_ConfigMultiProcessMode</li> <li>• CR2 CLKEN LL_USART_ConfigMultiProcessMode</li> <li>• CR3 SCEN LL_USART_ConfigMultiProcessMode</li> <li>• CR3 HDSEL LL_USART_ConfigMultiProcessMode</li> <li>• CR3 IREN LL_USART_ConfigMultiProcessMode</li> </ul>

**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:  
• ISR 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:  
• ISR 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:  
• ISR 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:  
• ISR 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:  
• ISR 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:  
• ISR 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:  
• ISR 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:  
• ISR TXE LL\_USART\_IsActiveFlag\_TXE

**LL\_USART\_IsActiveFlag\_LBD**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD(     USART_TypeDef * USARTx)</code>
Function Description	Check if the USART LIN Break Detection Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR LBDF LL_USART_IsActiveFlag_LBD</li> </ul>

**LL\_USART\_IsActiveFlag\_nCTS**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS(     USART_TypeDef * USARTx)</code>
Function Description	Check if the USART CTS interrupt Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR CTSIF LL_USART_IsActiveFlag_nCTS</li> </ul>

**LL\_USART\_IsActiveFlag\_CTS**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS(     USART_TypeDef * USARTx)</code>
Function Description	Check if the USART CTS Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR CTS LL_USART_IsActiveFlag_CTS</li> </ul>

**LL\_USART\_IsActiveFlag\_RTO**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO(     USART_TypeDef * USARTx)</code>
---------------	--

Function Description	Check if the USART Receiver Time Out Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR RTOF LL_USART_IsActiveFlag_RTO</li> </ul>

### LL\_USART\_IsActiveFlag\_EOB

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (USART_TypeDef * USARTx)</code>
Function Description	Check if the USART End Of Block Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR EOBF LL_USART_IsActiveFlag_EOB</li> </ul>

### LL\_USART\_IsActiveFlag\_ABRE

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (USART_TypeDef * USARTx)</code>
Function Description	Check if the USART Auto-Baud Rate Error Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR ABRE LL_USART_IsActiveFlag_ABRE</li> </ul>

### LL\_USART\_IsActiveFlag\_ABR

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (USART_TypeDef * USARTx)</code>
Function Description	Check if the USART Auto-Baud Rate Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx)</li> </ul>

RTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRF LL\_USART\_IsActiveFlag\_ABR

### **LL\_USART\_IsActiveFlag\_BUSY**

Function Name **\_STATIC\_INLINE uint32\_t LL\_USART\_IsActiveFlag\_BUSY (USART\_TypeDef \* USARTx)**

Function Description Check if the USART Busy Flag is set or not.

Parameters • **USARTx**: USART Instance

Return values • **State**: of bit (1 or 0).

Reference Manual to LL API cross reference:  
• ISR BUSY LL\_USART\_IsActiveFlag\_BUSY

### **LL\_USART\_IsActiveFlag\_CM**

Function Name **\_STATIC\_INLINE uint32\_t LL\_USART\_IsActiveFlag\_CM (USART\_TypeDef \* USARTx)**

Function Description Check if the USART Character Match Flag is set or not.

Parameters • **USARTx**: USART Instance

Return values • **State**: of bit (1 or 0).

Reference Manual to LL API cross reference:  
• ISR CMF LL\_USART\_IsActiveFlag\_CM

### **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:  
• ISR SBKF 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<br>Reference Manual to LL API cross reference: | <ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> <li>• ISR RWU LL_USART_IsActiveFlag_RWU</li> </ul> |
|--|---|

### **LL\_USART\_IsActiveFlag\_WKUP**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP(USART_TypeDef * USARTx)</code></b>
Function Description	Check if the USART Wake Up from stop mode Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx)</code> can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR WUF LL_USART_IsActiveFlag_WKUP</li> </ul>

### **LL\_USART\_IsActiveFlag\_TEACK**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK(USART_TypeDef * USARTx)</code></b>
Function Description	Check if the USART Transmit Enable Acknowledge Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR TEACK LL_USART_IsActiveFlag_TEACK</li> </ul>

### **LL\_USART\_IsActiveFlag\_REACK**

Function Name	<b><code>__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK(USART_TypeDef * USARTx)</code></b>
Function Description	Check if the USART Receive Enable Acknowledge Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ISR REACK LL_USART_IsActiveFlag_REACK</li> </ul>

### **LL\_USART\_ClearFlag\_PE**

Function Name	<b><code>__STATIC_INLINE void LL_USART_ClearFlag_PE</code></b>
---------------	--

**(USART\_TypeDef \* USARTx)**

Function Description	Clear Parity Error Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR PECF LL_USART_ClearFlag_PE</li> </ul>

**LL\_USART\_ClearFlag\_FE**

Function Name      **\_STATIC\_INLINE void LL\_USART\_ClearFlag\_FE  
(USART\_TypeDef \* USARTx)**

Function Description	Clear Framing Error Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR FECF LL_USART_ClearFlag_FE</li> </ul>

**LL\_USART\_ClearFlag\_NE**

Function Name      **\_STATIC\_INLINE void LL\_USART\_ClearFlag\_NE  
(USART\_TypeDef \* USARTx)**

Function Description	Clear Noise detected Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR NCF LL_USART_ClearFlag_NE</li> </ul>

**LL\_USART\_ClearFlag\_ORE**

Function Name      **\_STATIC\_INLINE void LL\_USART\_ClearFlag\_ORE  
(USART\_TypeDef \* USARTx)**

Function Description	Clear OverRun Error Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR ORECF LL_USART_ClearFlag_ORE</li> </ul>

**LL\_USART\_ClearFlag\_IDLE**

Function Name      **\_STATIC\_INLINE void LL\_USART\_ClearFlag\_IDLE  
(USART\_TypeDef \* USARTx)**

---

Function Description	Clear IDLE line detected Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR IDLECF LL_USART_ClearFlag_IDLE</li> </ul>

### LL\_USART\_ClearFlag\_TC

Function Name	<b><code>__STATIC_INLINE void LL_USART_ClearFlag_TC( USART_TypeDef * USARTx)</code></b>
Function Description	Clear Transmission Complete Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR TCCF LL_USART_ClearFlag_TC</li> </ul>

### LL\_USART\_ClearFlag\_LBD

Function Name	<b><code>__STATIC_INLINE void LL_USART_ClearFlag_LBD( USART_TypeDef * USARTx)</code></b>
Function Description	Clear LIN Break Detection Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR LBDCF LL_USART_ClearFlag_LBD</li> </ul>

### LL\_USART\_ClearFlag\_nCTS

Function Name	<b><code>__STATIC_INLINE void LL_USART_ClearFlag_nCTS( USART_TypeDef * USARTx)</code></b>
Function Description	Clear CTS Interrupt Flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• ICR CTSCF LL_USART_ClearFlag_nCTS</li> </ul>

**LL\_USART\_ClearFlag\_RTO**

Function Name      **`_STATIC_INLINE void LL_USART_ClearFlag_RTO  
(USART_TypeDef * USARTx)`**

Function Description      Clear Receiver Time Out Flag.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
• ICR RTOCF LL\_USART\_ClearFlag\_RTO

**LL\_USART\_ClearFlag\_EOB**

Function Name      **`_STATIC_INLINE void LL_USART_ClearFlag_EOB  
(USART_TypeDef * USARTx)`**

Function Description      Clear End Of Block Flag.

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:  
• ICR EOBCF LL\_USART\_ClearFlag\_EOB

**LL\_USART\_ClearFlag\_CM**

Function Name      **`_STATIC_INLINE void LL_USART_ClearFlag_CM  
(USART_TypeDef * USARTx)`**

Function Description      Clear Character Match Flag.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
• ICR CMCF LL\_USART\_ClearFlag\_CM

**LL\_USART\_ClearFlag\_WKUP**

Function Name      **`_STATIC_INLINE void LL_USART_ClearFlag_WKUP  
(USART_TypeDef * USARTx)`**

Function Description      Clear Wake Up from stop mode Flag.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

Notes      • Macro IS\_UART\_WAKEUP\_FROMSTOP\_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode

feature is supported by the USARTx instance.

Reference Manual to  
LL API cross  
reference:

- ICR WUCF LL\_USART\_ClearFlag\_WKUP

### **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\_CM**

Function Name      **`_STATIC_INLINE void LL_USART_EnableIT_CM  
(USART_TypeDef * USARTx)`**

Function Description      Enable Character Match Interrupt.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR1 CMIE LL\_USART\_EnableIT\_CM

### **LL\_USART\_EnableIT\_RTO**

Function Name      **`_STATIC_INLINE void LL_USART_EnableIT_RTO  
(USART_TypeDef * USARTx)`**

Function Description      Enable Receiver Timeout Interrupt.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

- Reference Manual to  
LL API cross  
reference:
- CR1 RTOIE LL\_USART\_EnableIT\_RTO

### **LL\_USART\_EnableIT\_EOB**

Function Name      **`_STATIC_INLINE void LL_USART_EnableIT_EOB  
(USART_TypeDef * USARTx)`**

Function Description      Enable End Of Block Interrupt.

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:

- CR1 EOBIIE LL\_USART\_EnableIT\_EOB

### **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\_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx\_ISR 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

- CR3 CTSIE LL\_USART\_EnableIT\_CTS

LL API cross  
reference:

### **LL\_USART\_EnableIT\_WKUP**

Function Name	<b><code>_STATIC_INLINE void LL_USART_EnableIT_WKUP( (USART_TypeDef * USARTx)</code></b>
Function Description	Enable Wake Up from Stop Mode Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx)</code> can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 WUFIE LL_USART_EnableIT_WKUP</li> </ul>

### **LL\_USART\_DisableIT\_IDLE**

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableIT_IDLE( (USART_TypeDef * USARTx)</code></b>
Function Description	Disable IDLE Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 IDLEIE LL_USART_DisableIT_IDLE</li> </ul>

### **LL\_USART\_DisableIT\_RXNE**

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableIT_RXNE( (USART_TypeDef * USARTx)</code></b>
Function Description	Disable RX Not Empty Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 RXNEIE LL_USART_DisableIT_RXNE</li> </ul>

### **LL\_USART\_DisableIT\_TC**

Function Name	<b><code>_STATIC_INLINE void LL_USART_DisableIT_TC( (USART_TypeDef * USARTx)</code></b>
Function Description	Disable Transmission Complete Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>

---

Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 TCIE LL_USART_DisableIT_TC</li></ul>

### LL\_USART\_DisableIT\_TXE

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_TXE (USART_TypeDef * USARTx)</code>
Function Description	Disable TX Empty Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 TXEIE LL_USART_DisableIT_TXE</li></ul>

### LL\_USART\_DisableIT\_PE

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)</code>
Function Description	Disable Parity Error Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 PEIE LL_USART_DisableIT_PE</li></ul>

### LL\_USART\_DisableIT\_CM

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)</code>
Function Description	Disable Character Match Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR1 CMIE LL_USART_DisableIT_CM</li></ul>

### LL\_USART\_DisableIT\_RTO

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)</code>
Function Description	Disable Receiver Timeout Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>

- Reference Manual to  
LL API cross  
reference:
- CR1 RTOIE LL\_USART\_DisableIT\_RTO

### **LL\_USART\_DisableIT\_EOB**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_EOB(     USART_TypeDef * USARTx)</code>
Function Description	Disable End Of Block Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 EOBIIE LL_USART_DisableIT_EOB</li> </ul>

### **LL\_USART\_DisableIT\_LBD**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_LBD(     USART_TypeDef * USARTx)</code>
Function Description	Disable LIN Break Detection Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR2 LBDIE LL_USART_DisableIT_LBD</li> </ul>

### **LL\_USART\_DisableIT\_ERROR**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_ERROR(     USART_TypeDef * USARTx)</code>
Function Description	Disable Error Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.</li> </ul>
Reference Manual to LL API cross	<ul style="list-style-type: none"> <li>• CR3 EIE LL_USART_DisableIT_ERROR</li> </ul>

reference:

### **LL\_USART\_DisableIT\_CTS**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)</code>
Function Description	Disable CTS Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSIE LL_USART_DisableIT_CTS</li> </ul>

### **LL\_USART\_DisableIT\_WKUP**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)</code>
Function Description	Disable Wake Up from Stop Mode Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 WUFIE LL_USART_DisableIT_WKUP</li> </ul>

### **LL\_USART\_IsEnabledIT\_IDLE**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (USART_TypeDef * USARTx)</code>
Function Description	Check if the USART IDLE Interrupt source is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR1 IDLEIE LL_USART_IsEnabledIT_IDLE</li> </ul>

### **LL\_USART\_IsEnabledIT\_RXNE**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE (USART_TypeDef * USARTx)</code>
---------------	--

Function Description	Check if the USART RX Not Empty Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 RXNEIE LL_USART_IsEnabledIT_RXNE

### LL\_USART\_IsEnabledIT\_TC

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC(USART_TypeDef * USARTx)</code>
Function Description	Check if the USART Transmission Complete Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 TCIE LL_USART_IsEnabledIT_TC

### LL\_USART\_IsEnabledIT\_TXE

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE(USART_TypeDef * USARTx)</code>
Function Description	Check if the USART TX Empty Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 TXEIE LL_USART_IsEnabledIT_TXE

### LL\_USART\_IsEnabledIT\_PE

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE(USART_TypeDef * USARTx)</code>
Function Description	Check if the USART Parity Error Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 PEIE LL_USART_IsEnabledIT_PE

### LL\_USART\_IsEnabledIT\_CM

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM(USART_TypeDef * USARTx)</code>
---------------	---

Function Description	Check if the USART Character Match Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 CMIE LL_USART_IsEnabledIT_CM

### LL\_USART\_IsEnabledIT\_RTO

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO(USART_TypeDef * USARTx)</code>
Function Description	Check if the USART Receiver Timeout Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR1 RTOIE LL_USART_IsEnabledIT_RTO

### LL\_USART\_IsEnabledIT\_EOB

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB(USART_TypeDef * USARTx)</code>
Function Description	Check if the USART End Of Block Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	CR1 EOBIIE LL_USART_IsEnabledIT_EOB

### LL\_USART\_IsEnabledIT\_LBD

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD(USART_TypeDef * USARTx)</code>
Function Description	Check if the USART LIN Break Detection Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li><b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.</li> </ul>

- Reference Manual to  
LL API cross  
reference:
- CR2 LBDIE LL\_USART\_IsEnabledIT\_LBD

### **LL\_USART\_IsEnabledIT\_ERROR**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR(USART_TypeDef * USARTx)</code></b>
Function Description	Check if the USART Error Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 EIE LL_USART_IsEnabledIT_ERROR</li> </ul>

### **LL\_USART\_IsEnabledIT\_CTS**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS(USART_TypeDef * USARTx)</code></b>
Function Description	Check if the USART CTS Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 CTSIE LL_USART_IsEnabledIT_CTS</li> </ul>

### **LL\_USART\_IsEnabledIT\_WKUP**

Function Name	<b><code>_STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP(USART_TypeDef * USARTx)</code></b>
Function Description	Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR3 WUFIE LL_USART_IsEnabledIT_WKUP</li> </ul>

**LL\_USART\_EnableDMAReq\_RX**

Function Name	<code>__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)</code>
Function Description	Enable DMA Mode for reception.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR3 DMAR LL_USART_EnableDMAReq_RX</li></ul>

**LL\_USART\_DisableDMAReq\_RX**

Function Name	<code>__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)</code>
Function Description	Disable DMA Mode for reception.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR3 DMAR LL_USART_DisableDMAReq_RX</li></ul>

**LL\_USART\_IsEnabledDMAReq\_RX**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (USART_TypeDef * USARTx)</code>
Function Description	Check if DMA Mode is enabled for reception.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>State:</b> of bit (1 or 0).</li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR3 DMAR LL_USART_IsEnabledDMAReq_RX</li></ul>

**LL\_USART\_EnableDMAReq\_TX**

Function Name	<code>__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)</code>
Function Description	Enable DMA Mode for transmission.
Parameters	<ul style="list-style-type: none"><li>• <b>USARTx:</b> USART Instance</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None:</b></li></ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"><li>• CR3 DMAT LL_USART_EnableDMAReq_TX</li></ul>

**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\_EnableDMADeactOnRxErr**

Function Name      **STATIC\_INLINE void LL\_USART\_EnableDMADeactOnRxErr  
(USART\_TypeDef \* USARTx)**

Function Description      Enable DMA Disabling on Reception Error.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
CR3 DDRE LL\_USART\_EnableDMADeactOnRxErr

**LL\_USART\_DisableDMADeactOnRxErr**

Function Name      **STATIC\_INLINE void LL\_USART\_DisableDMADeactOnRxErr  
(USART\_TypeDef \* USARTx)**

Function Description      Disable DMA Disabling on Reception Error.

Parameters      • **USARTx:** USART Instance

Return values      • **None:**

Reference Manual to  
LL API cross  
reference:  
CR3 DDRE LL\_USART\_DisableDMADeactOnRxErr

**LL\_USART\_IsEnabledDMADeactOnRxErr**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (USART_TypeDef * USARTx)</code>
Function Description	Indicate if DMA Disabling on Reception Error is disabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

**LL\_USART\_DMA\_GetRegAddr**

Function Name	<code>__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (USART_TypeDef * USARTx, uint32_t Direction)</code>
Function Description	Get the data register address used for DMA transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Direction:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- LL_USART_DMA_REG_DATA_TRANSMIT</li> <li>- LL_USART_DMA_REG_DATA_RECEIVE</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Address:</b> of data register</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RDR RDR LL_USART_DMA_GetRegAddr</li> <li>• TDR TDR LL_USART_DMA_GetRegAddr</li> </ul>

**LL\_USART\_ReceiveData8**

Function Name	<code>__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (USART_TypeDef * USARTx)</code>
Function Description	Read Receiver Data register (Receive Data value, 8 bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Reference Manual to LL API cross reference:	• RDR RDR LL_USART_ReceiveData8

**LL\_USART\_ReceiveData9**

Function Name	<code>__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (USART_TypeDef * USARTx)</code>
Function Description	Read Receiver Data register (Receive Data value, 9 bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x1FF</li> </ul>
Reference Manual to	• RDR RDR LL_USART_ReceiveData9

LL API cross  
reference:

### **LL\_USART\_TransmitData8**

Function Name	<code>__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)</code>
Function Description	Write in Transmitter Data Register (Transmit Data value, 8 bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0xFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TDR TDR LL_USART_TransmitData8</li> </ul>

### **LL\_USART\_TransmitData9**

Function Name	<code>__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)</code>
Function Description	Write in Transmitter Data Register (Transmit Data value, 9 bits)
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>Value:</b> between Min_Data=0x00 and Max_Data=0x1FF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• TDR TDR LL_USART_TransmitData9</li> </ul>

### **LL\_USART\_RequestAutoBaudRate**

Function Name	<code>__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)</code>
Function Description	Request an Automatic Baud Rate measurement on next received data frame.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro <code>IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx)</code> can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• RQR ABRRQ LL_USART_RequestAutoBaudRate</li> </ul>

### **LL\_USART\_RequestBreakSending**

Function Name	<code>__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)</code>
---------------	---

---

Function Description	Request Break sending.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	• RQR SBKRQ LL_USART_RequestBreakSending

### LL\_USART\_RequestEnterMuteMode

Function Name	<b><code>_STATIC_INLINE void LL_USART_RequestEnterMuteMode(USART_TypeDef * USARTx)</code></b>
Function Description	Put USART in mute mode and set the RWU flag.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	• RQR MMRQ LL_USART_RequestEnterMuteMode

### LL\_USART\_RequestRxDataFlush

Function Name	<b><code>_STATIC_INLINE void LL_USART_RequestRxDataFlush(USART_TypeDef * USARTx)</code></b>
Function Description	Request a Receive Data flush.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Reference Manual to LL API cross reference:	• RQR RXFRQ LL_USART_RequestRxDataFlush

### LL\_USART\_RequestTxDataFlush

Function Name	<b><code>_STATIC_INLINE void LL_USART_RequestTxDataFlush(USART_TypeDef * USARTx)</code></b>
Function Description	Request a Transmit data flush.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.</li> </ul>
Reference Manual to LL API cross reference:	• RQR TXFRQ LL_USART_RequestTxDataFlush

**LL\_USART\_DelInit**

Function Name	<b>ErrorStatus LL_USART_DelInit (USART_TypeDef * USARTx)</b>
Function Description	De-initialize USART registers (Registers restored to their default values).
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: USART registers are de-initialized</li> <li>– ERROR: USART registers are not de-initialized</li> </ul> </li> </ul>

**LL\_USART\_Init**

Function Name	<b>ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, LL_USART_InitTypeDef * USART_InitStruct)</b>
Function Description	Initialize USART registers according to the specified parameters in USART_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> <li>• <b>USART_InitStruct:</b> pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>An:</b> ErrorStatus enumeration value:           <ul style="list-style-type: none"> <li>– SUCCESS: USART registers are initialized according to USART_InitStruct content</li> <li>– ERROR: Problem occurred during USART Registers initialization</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).</li> </ul>

**LL\_USART\_StructInit**

Function Name	<b>void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)</b>
Function Description	Set each LL_USART_InitTypeDef field to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>USART_InitStruct:</b> pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

**LL\_USART\_ClockInit**

Function Name	<b>ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, LL_USART_ClockInitTypeDef * USART_ClockInitStruct)</b>
Function Description	Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>USARTx:</b> USART Instance</li> </ul>

- **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	<b>void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)</b>
Function Description	Set each field of a LL_USART_ClockInitTypeDef type structure to default value.
Parameters	<ul style="list-style-type: none"> <li>• <b>USART_ClockInitStruct:</b> pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>

## **75.3 USART Firmware driver defines**

### **75.3.1 USART**

#### ***Address Length Detection***

LL_USART_ADDRESS_DETECT_4B	4-bit address detection method selected
LL_USART_ADDRESS_DETECT_7B	7-bit address detection (in 8-bit data mode) method selected

#### ***Autobaud Detection***

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT	Measurement of the start bit is used to detect the baud rate
LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE	Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start1xxxxxxxx
LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME	0x7F frame detection
LL_USART_AUTOBAUD_DETECT_ON_55_FRAME	0x55 frame detection

#### ***Binary Data Inversion***

LL_USART_BINARY_LOGIC_POSITIVE	Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
--------------------------------	--

**LL\_USART\_BINARY\_LOGIC\_NEGATIVE** Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

#### **Bit Order**

**LL\_USART\_BITORDER\_LSBFIRST** data is transmitted/received with data bit 0 first, following the start bit

**LL\_USART\_BITORDER\_MSBFIRST** data is transmitted/received with the MSB first, following the start bit

#### **Clear Flags Defines**

<b>LL_USART_ICR_PECF</b>	Parity error flag
<b>LL_USART_ICR_FEFC</b>	Framing error flag
<b>LL_USART_ICR_NCF</b>	Noise detected flag
<b>LL_USART_ICR_ORECF</b>	Overrun error flag
<b>LL_USART_ICR_IDLECF</b>	Idle line detected flag
<b>LL_USART_ICR_TCCF</b>	Transmission complete flag
<b>LL_USART_ICR_LBDCF</b>	LIN break detection flag
<b>LL_USART_ICR_CTSCF</b>	CTS flag
<b>LL_USART_ICR_RTOCF</b>	Receiver timeout flag
<b>LL_USART_ICR_EOBCF</b>	End of block flag
<b>LL_USART_ICR_CMCF</b>	Character match flag
<b>LL_USART_ICR_WUCF</b>	Wakeup from Stop mode flag

#### **Clock Signal**

**LL\_USART\_CLOCK\_DISABLE** Clock signal not provided

**LL\_USART\_CLOCK\_ENABLE** Clock signal provided

#### **Datawidth**

**LL\_USART\_DATAWIDTH\_7B** 7 bits word length : Start bit, 7 data bits, n stop bits

**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

#### **Driver Enable Polarity**

**LL\_USART\_DE\_POLARITY\_HIGH** DE signal is active high

**LL\_USART\_DE\_POLARITY\_LOW** DE signal is active low

#### **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

#### **DMA Register Data**

**LL\_USART\_DMA\_REG\_DATA\_TRANSMIT** Get address of data register used for

LL_USART_DMA_REG_DATA_RECEIVE	Get address of data register used for reception
	transmission

**Get Flags Defines**

LL_USART_ISR_PE	Parity error flag
LL_USART_ISR_FE	Framing error flag
LL_USART_ISR_NE	Noise detected flag
LL_USART_ISR_ORE	Overrun error flag
LL_USART_ISR_IDLE	Idle line detected flag
LL_USART_ISR_RXNE	Read data register not empty flag
LL_USART_ISR_TC	Transmission complete flag
LL_USART_ISR_TXE	Transmit data register empty flag
LL_USART_ISR_LBDF	LIN break detection flag
LL_USART_ISR_CTSIF	CTS interrupt flag
LL_USART_ISR_CTS	CTS flag
LL_USART_ISR_RTOF	Receiver timeout flag
LL_USART_ISR_EOBF	End of block flag
LL_USART_ISR_ABRE	Auto baud rate error flag
LL_USART_ISR_ABRF	Auto baud rate flag
LL_USART_ISR_BUSY	Busy flag
LL_USART_ISR_CMF	Character match flag
LL_USART_ISR_SBKF	Send break flag
LL_USART_ISR_RWU	Receiver wakeup from Mute mode flag
LL_USART_ISR_WUF	Wakeup from Stop mode flag
LL_USART_ISR_TEACK	Transmit enable acknowledge flag
LL_USART_ISR_REACK	Receive enable acknowledge 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_CR1_CMIE	Character match interrupt enable
LL_USART_CR1_RTOIE	Receiver timeout interrupt enable
LL_USART_CR1_EOBIE	End of Block interrupt enable
LL_USART_CR2_LBDIE	LIN break detection interrupt enable
LL_USART_CR3_EIE	Error interrupt enable
LL_USART_CR3_CTSIE	CTS interrupt enable
LL_USART_CR3_WUFIE	Wakeup from Stop mode 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

#### **Oversampling**

LL_USART_OVERSAMPLING_16	Oversampling by 16
LL_USART_OVERSAMPLING_8	Oversampling by 8

#### **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

#### **RX Pin Active Level Inversion**

LL_USART_RXPIN_LEVEL_STANDARD	RX pin signal works using the standard logic levels
-------------------------------	---

`LL_USART_RXPIN_LEVEL_INVERTED` RX pin signal values are inverted.

#### **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

#### **TX Pin Active Level Inversion**

`LL_USART_TXPIN_LEVEL_STANDARD` TX pin signal works using the standard logic levels

`LL_USART_TXPIN_LEVEL_INVERTED` TX pin signal values are inverted.

#### **TX RX Pins Swap**

`LL_USART_TXRX_STANDARD` TX/RX pins are used as defined in standard pinout

`LL_USART_TXRX_SWAPPED` TX and RX pins functions are swapped.

#### **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

#### **Wakeup Activation**

`LL_USART_WAKEUP_ON_ADDRESS` Wake up active on address match

`LL_USART_WAKEUP_ON_STARTBIT` Wake up active on Start bit detection

`LL_USART_WAKEUP_ON_RXNE` Wake up active on RXNE

#### **Exported Macros Helper**

`_LL_USART_DIV_SAMPLING8` **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_DIV_SAMPLING16` **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

**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

## 76 LL UTILS Generic Driver

### 76.1 UTILS Firmware driver registers structures

#### 76.1.1 LL\_UTILS\_PLLInitTypeDef

##### Data Fields

- *uint32\_t PLLMul*
- *uint32\_t PLLDiv*

##### 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::PLLDiv*  
Division factor for PLL VCO output clock. This parameter can be a value of [RCC\\_LL\\_EC\\_PLL\\_DIV](#)This feature can be modified afterwards using unitary function [LL\\_RCC\\_PLL\\_ConfigDomain\\_SYS\(\)](#).

#### 76.1.2 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\(\)](#).

## 76.2 UTILS Firmware driver API description

### 76.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 32000000 Hz.

This section contains the following APIs:

- [\*\*LL\\_SetSystemCoreClock\(\)\*\*](#)
- [\*\*LL\\_PLL\\_ConfigSystemClock\\_HSI\(\)\*\*](#)
- [\*\*LL\\_PLL\\_ConfigSystemClock\\_HSE\(\)\*\*](#)

### 76.2.2 Detailed description of functions

#### LL\_GetUID\_Word0

Function Name	<code>__STATIC_INLINE uint32_t LL_GetUID_Word0 (void )</code>
Function Description	Get Word0 of the unique device identifier (UID based on 96 bits)
Return values	<ul style="list-style-type: none"> <li><b>UID[31:0]:</b></li> </ul>

#### LL\_GetUID\_Word1

Function Name	<code>__STATIC_INLINE uint32_t LL_GetUID_Word1 (void )</code>
Function Description	Get Word1 of the unique device identifier (UID based on 96 bits)
Return values	<ul style="list-style-type: none"> <li><b>UID[63:32]:</b></li> </ul>

#### LL\_GetUID\_Word2

Function Name	<code>__STATIC_INLINE uint32_t LL_GetUID_Word2 (void )</code>
Function Description	Get Word2 of the unique device identifier (UID based on 96 bits)
Return values	<ul style="list-style-type: none"> <li><b>UID[95:64]:</b></li> </ul>

#### LL\_GetFlashSize

Function Name	<code>__STATIC_INLINE uint32_t LL_GetFlashSize (void )</code>
Function Description	Get Flash memory size.
Return values	<ul style="list-style-type: none"> <li><b>FLASH_SIZE[15:0]:</b> Flash memory size</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.</li> </ul>

#### LL\_InitTick

Function Name	<code>__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)</code>
Function Description	This function configures the Cortex-M SysTick source of the time base.
Parameters	<ul style="list-style-type: none"> <li><b>HCLKFrequency:</b> HCLK frequency in Hz (can be calculated</li> </ul>

	thanks to RCC helper macro)
Return values	• <b>Ticks:</b> Number of ticks
Notes	• <b>None:</b>
	• 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	<b>void LL_Init1msTick (uint32_t HCLKFrequency)</b>
Function Description	This function configures the Cortex-M SysTick source to have 1ms time base.
Parameters	• <b>HCLKFrequency:</b> HCLK frequency in Hz
Return values	• <b>None:</b>
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\_mDdelay

Function Name	<b>void LL_mDdelay (uint32_t Delay)</b>
Function Description	This function provides accurate delay (in milliseconds) based on SysTick counter flag.
Parameters	• <b>Delay:</b> specifies the delay time length, in milliseconds.
Return values	• <b>None:</b>
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	<b>void LL_SetSystemCoreClock (uint32_t HCLKFrequency)</b>
Function Description	This function sets directly SystemCoreClock CMSIS variable.
Parameters	• <b>HCLKFrequency:</b> HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
Return values	• <b>None:</b>
Notes	• Variable can be calculated also through SystemCoreClockUpdate function.

### LL\_PLL\_ConfigSystemClock\_HSI

Function Name	<b>ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct,</b>
---------------	---

**LL\_UTILS\_ClkInitTypeDef \* UTILS\_ClkInitStruct)**

Function Description	This function configures system clock with HSI as clock source of the PLL.
Parameters	<ul style="list-style-type: none"> <li><b>UTILS_PLLInitStruct:</b> pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.</li> <li><b>UTILS_ClkInitStruct:</b> pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>SUCCESS: Max frequency configuration done</li> <li>ERROR: Max frequency configuration not done</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>The application need to ensure that PLL is disabled.</li> <li>Function is based on the following formula: PLL output frequency = ((HSI frequency * PLLMul) / PLLDiv)PLLMul: The application software must set correctly the PLL multiplication factor to avoid exceeding 96 MHz as PLLVCO when the product is in range 1,48 MHz as PLLVCO when the product is in range 2,24 MHz when the product is in range 3</li> <li>FLASH latency can be modified through this function.</li> </ul>

**LL\_PLL\_ConfigSystemClock\_HSE**

Function Name	<b>ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)</b>
Function Description	This function configures system clock with HSE as clock source of the PLL.
Parameters	<ul style="list-style-type: none"> <li><b>HSEFrequency:</b> Value between Min_Data = 1000000 and Max_Data = 24000000</li> <li><b>HSEBypass:</b> This parameter can be one of the following values: <ul style="list-style-type: none"> <li>LL_UTILS_HSEBYPASS_ON</li> <li>LL_UTILS_HSEBYPASS_OFF</li> </ul> </li> <li><b>UTILS_PLLInitStruct:</b> pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.</li> <li><b>UTILS_ClkInitStruct:</b> pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>An:</b> ErrorStatus enumeration value: <ul style="list-style-type: none"> <li>SUCCESS: Max frequency configuration done</li> <li>ERROR: Max frequency configuration not done</li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>The application need to ensure that PLL is disabled.</li> <li>Function is based on the following formula: PLL output frequency = ((HSE frequency * PLLMul) / PLLDiv)PLLMul: The application software must set correctly the PLL multiplication factor to avoid exceeding 96 MHz as PLLVCO when the product is in range 1,48 MHz as PLLVCO when the product is in range 2,24 MHz when the product is in range 3</li> </ul>

- product is in range 2,24 MHz when the product is in range 3
- FLASH latency can be modified through this function.

## 76.3 UTILS Firmware driver defines

### 76.3.1 UTILS

#### *HSE Bypass activation*

LL\_UTILS\_HSEBYPASS\_OFF HSE Bypass is not enabled

LL\_UTILS\_HSEBYPASS\_ON HSE Bypass is enabled

## 77 LL WWDG Generic Driver

### 77.1 WWDG Firmware driver API description

#### 77.1.1 Detailed description of functions

##### **LL\_WWDG\_Enable**

Function Name	<code>__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)</code>
Function Description	Enable Window Watchdog.
Parameters	<ul style="list-style-type: none"> <li>• <b>WWDGx:</b> WWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WDGA LL_WWDG_Enable</li> </ul>

##### **LL\_WWDG\_IsEnabled**

Function Name	<code>__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)</code>
Function Description	Checks if Window Watchdog is enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>WWDGx:</b> WWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CR WDGA LL_WWDG_IsEnabled</li> </ul>

##### **LL\_WWDG\_SetCounter**

Function Name	<code>__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)</code>
Function Description	Set the Watchdog counter value to provided value (7-bits T[6:0])
Parameters	<ul style="list-style-type: none"> <li>• <b>WWDGx:</b> WWDG Instance</li> <li>• <b>Counter:</b> 0..0x7F (7 bit counter value)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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 2<sup>exp(WDGTB)</sup>) PCLK cycles. A reset is produced when it rolls over from 0x40 to 0x3F (bit T6</li> </ul>

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 2<sup>expWDGTB</sup>) 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
  - CFR WDGTB LL\_WWDG\_GetPrescaler
- Reference Manual to  
LL API cross  
reference:

### LL\_WWDG\_SetWindow

Function Name	<code>__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)</code>
Function Description	Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).
Parameters	<ul style="list-style-type: none"> <li>• <b>WWDGx:</b> WWDG Instance</li> <li>• <b>Window:</b> 0x00..0x7F (7 bit Window value)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>None:</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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 than 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFR W LL_WWDG_SetWindow</li> </ul>

### LL\_WWDG\_GetWindow

Function Name	<code>__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)</code>
Function Description	Return current Watchdog Window Value (7 bits value)
Parameters	<ul style="list-style-type: none"> <li>• <b>WWDGx:</b> WWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>7:</b> bit Watchdog Window value</li> </ul>
Reference Manual to LL API cross reference:	<ul style="list-style-type: none"> <li>• CFR W LL_WWDG_SetWindow</li> </ul>

### LL\_WWDG\_IsActiveFlag\_EWKUP

Function Name	<code>__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)</code>
Function Description	Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.
Parameters	<ul style="list-style-type: none"> <li>• <b>WWDGx:</b> WWDG Instance</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>State:</b> of bit (1 or 0).</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This bit is set by hardware when the counter has reached the</li> </ul>

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

## 77.2 WWDG Firmware driver defines

### 77.2.1 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

## 78 Correspondence between API registers and API low-layer driver functions

### 78.1 ADC

Table 25: Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
CALFACT	CALFACT	<i>LL_ADC_GetCalibrationFactor</i>
		<i>LL_ADC_SetCalibrationFactor</i>
CCR	LFMEN	<i>LL_ADC_GetCommonFrequencyMode</i>
		<i>LL_ADC_SetCommonFrequencyMode</i>
CCR	PRESC	<i>LL_ADC_GetCommonClock</i>
		<i>LL_ADC_SetCommonClock</i>
CCR	TSEN	<i>LL_ADC_GetCommonPathInternalCh</i>
		<i>LL_ADC_SetCommonPathInternalCh</i>
CCR	VLCDEN	<i>LL_ADC_GetCommonPathInternalCh</i>
		<i>LL_ADC_SetCommonPathInternalCh</i>
CCR	VREFEN	<i>LL_ADC_GetCommonPathInternalCh</i>
		<i>LL_ADC_SetCommonPathInternalCh</i>
CFGREG1	ALIGN	<i>LL_ADC_GetDataAlignment</i>
		<i>LL_ADC_SetDataAlignment</i>
CFGREG1	AUTOFF	<i>LL_ADC_GetLowPowerMode</i>
		<i>LL_ADC_SetLowPowerMode</i>
CFGREG1	AWDCH	<i>LL_ADC_GetAnalogWDMonitChannels</i>
		<i>LL_ADC_SetAnalogWDMonitChannels</i>
CFGREG1	AWDEN	<i>LL_ADC_GetAnalogWDMonitChannels</i>
		<i>LL_ADC_SetAnalogWDMonitChannels</i>
CFGREG1	AWDSGL	<i>LL_ADC_GetAnalogWDMonitChannels</i>
		<i>LL_ADC_SetAnalogWDMonitChannels</i>
CFGREG1	CONT	<i>LL_ADC_REG_GetContinuousMode</i>
		<i>LL_ADC_REG_SetContinuousMode</i>
CFGREG1	DISCEN	<i>LL_ADC_REG_GetSequencerDiscont</i>
		<i>LL_ADC_REG_SetSequencerDiscont</i>
CFGREG1	DMACFG	<i>LL_ADC_REG_GetDMATransfer</i>
		<i>LL_ADC_REG_SetDMATransfer</i>
CFGREG1	DMAEN	<i>LL_ADC_REG_GetDMATransfer</i>
		<i>LL_ADC_REG_SetDMATransfer</i>

Register	Field	Function
	EXTEN	<i>LL_ADC_REG_GetTriggerEdge</i>
		<i>LL_ADC_REG_GetTriggerSource</i>
		<i>LL_ADC_REG_IsTriggerSourceSWStart</i>
		<i>LL_ADC_REG_SetTriggerEdge</i>
		<i>LL_ADC_REG_SetTriggerSource</i>
	EXTSEL	<i>LL_ADC_REG_GetTriggerSource</i>
		<i>LL_ADC_REG_SetTriggerSource</i>
	OVRMOD	<i>LL_ADC_REG_GetOverrun</i>
		<i>LL_ADC_REG_SetOverrun</i>
	RES	<i>LL_ADC_GetResolution</i>
		<i>LL_ADC_SetResolution</i>
	SCANDIR	<i>LL_ADC_REG_GetSequencerScanDirection</i>
		<i>LL_ADC_REG_SetSequencerScanDirection</i>
	WAIT	<i>LL_ADC_GetLowPowerMode</i>
		<i>LL_ADC_SetLowPowerMode</i>
CFGREG2	CKMODE	<i>LL_ADC_GetClock</i>
		<i>LL_ADC_SetClock</i>
	OVSE	<i>LL_ADC_GetOverSamplingScope</i>
		<i>LL_ADC_SetOverSamplingScope</i>
	OVSR	<i>LL_ADC_ConfigOverSamplingRatioShift</i>
		<i>LL_ADC_GetOverSamplingRatio</i>
	OVSS	<i>LL_ADC_ConfigOverSamplingRatioShift</i>
		<i>LL_ADC_GetOverSamplingShift</i>
	TOVS	<i>LL_ADC_GetOverSamplingDiscont</i>
		<i>LL_ADC_SetOverSamplingDiscont</i>
CHSELREG	CHSEL0	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL1	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL10	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>

Register	Field	Function
CHSEL11		<i>LL_ADC_REG_SetSequencerChannels</i>
		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL12		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL13		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL14		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL15		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL16		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL17		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL18		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CHSEL2		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>

Register	Field	Function
	CHSEL3	<i>LL_ADC_REG_SetSequencerChannels</i>
		<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL4	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL5	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL6	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL7	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL8	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
	CHSEL9	<i>LL_ADC_REG_GetSequencerChannels</i>
		<i>LL_ADC_REG_SetSequencerChAdd</i>
		<i>LL_ADC_REG_SetSequencerChRem</i>
		<i>LL_ADC_REG_SetSequencerChannels</i>
CR	ADCAL	<i>LL_ADC_IsCalibrationOnGoing</i>
		<i>LL_ADC_StartCalibration</i>
	ADDIS	<i>LL_ADC_Disable</i>
		<i>LL_ADC_IsDisableOngoing</i>
	ADEN	<i>LL_ADC_Enable</i>
		<i>LL_ADC_IsEnabled</i>
	ADSTART	<i>LL_ADC_REG_IsConversionOngoing</i>

Register	Field	Function
	ADSTP	<i>LL_ADC_REG_StartConversion</i>
		<i>LL_ADC_REG_IsStopConversionOngoing</i>
		<i>LL_ADC_REG_StopConversion</i>
	ADVREGEN	<i>LL_ADC_DisableInternalRegulator</i>
		<i>LL_ADC_EnableInternalRegulator</i>
		<i>LL_ADC_IsInternalRegulatorEnabled</i>
DR	DATA	<i>LL_ADC_DMA_GetRegAddr</i>
		<i>LL_ADC_REG_ReadConversionData10</i>
		<i>LL_ADC_REG_ReadConversionData12</i>
		<i>LL_ADC_REG_ReadConversionData32</i>
		<i>LL_ADC_REG_ReadConversionData6</i>
		<i>LL_ADC_REG_ReadConversionData8</i>
IER	ADRDYIE	<i>LL_ADC_DisableIT_ADRDY</i>
		<i>LL_ADC_EnableIT_ADRDY</i>
		<i>LL_ADC_IsEnabledIT_ADRDY</i>
	AWDIE	<i>LL_ADC_DisableIT_AWD1</i>
		<i>LL_ADC_EnableIT_AWD1</i>
		<i>LL_ADC_IsEnabledIT_AWD1</i>
	EOCALIE	<i>LL_ADC_DisableIT_EOCAL</i>
		<i>LL_ADC_EnableIT_EOCAL</i>
		<i>LL_ADC_IsEnabledIT_EOCAL</i>
	EOCIE	<i>LL_ADC_DisableIT_EOC</i>
		<i>LL_ADC_EnableIT_EOC</i>
		<i>LL_ADC_IsEnabledIT_EOC</i>
	EOSEQIE	<i>LL_ADC_DisableIT_EOS</i>
		<i>LL_ADC_EnableIT_EOS</i>
		<i>LL_ADC_IsEnabledIT_EOS</i>
	EOSMPIE	<i>LL_ADC_DisableIT_EOSMP</i>
		<i>LL_ADC_EnableIT_EOSMP</i>
		<i>LL_ADC_IsEnabledIT_EOSMP</i>
	OVRIE	<i>LL_ADC_DisableIT_OVR</i>
		<i>LL_ADC_EnableIT_OVR</i>
		<i>LL_ADC_IsEnabledIT_OVR</i>
ISR	ADRDY	<i>LL_ADC_ClearFlag_ADRDY</i>
		<i>LL_ADC_IsActiveFlag_ADRDY</i>
	AWD	<i>LL_ADC_ClearFlag_AWD1</i>

Register	Field	Function
	EOC	<code>LL_ADC_IsActiveFlag_AWD1</code>
		<code>LL_ADC_ClearFlag_EOC</code>
		<code>LL_ADC_IsActiveFlag_EOC</code>
	EOCAL	<code>LL_ADC_ClearFlag_EOCAL</code>
		<code>LL_ADC_IsActiveFlag_EOCAL</code>
	EOSEQ	<code>LL_ADC_ClearFlag_EOS</code>
		<code>LL_ADC_IsActiveFlag_EOS</code>
	EOSMP	<code>LL_ADC_ClearFlag_EOSMP</code>
		<code>LL_ADC_IsActiveFlag_EOSMP</code>
	OVR	<code>LL_ADC_ClearFlag_OVR</code>
		<code>LL_ADC_IsActiveFlag_OVR</code>
SMPR	SMP	<code>LL_ADC_GetSamplingTimeCommonChannels</code>
		<code>LL_ADC_SetSamplingTimeCommonChannels</code>
TR	HT	<code>LL_ADC_ConfigAnalogWDThresholds</code>
		<code>LL_ADC_GetAnalogWDThresholds</code>
		<code>LL_ADC_SetAnalogWDThresholds</code>
	LT	<code>LL_ADC_ConfigAnalogWDThresholds</code>
		<code>LL_ADC_GetAnalogWDThresholds</code>
		<code>LL_ADC_SetAnalogWDThresholds</code>

## 78.2 BUS

Table 26: Correspondence between BUS registers and BUS low-layer driver functions

Register	Field	Function
AHBENR	CRCEN	<code>LL_AHB1_GRP1_DisableClock</code>
		<code>LL_AHB1_GRP1_EnableClock</code>
		<code>LL_AHB1_GRP1_IsEnabledClock</code>
	CRYPEN	<code>LL_AHB1_GRP1_DisableClock</code>
		<code>LL_AHB1_GRP1_EnableClock</code>
		<code>LL_AHB1_GRP1_IsEnabledClock</code>
	DMAEN	<code>LL_AHB1_GRP1_DisableClock</code>
		<code>LL_AHB1_GRP1_EnableClock</code>
		<code>LL_AHB1_GRP1_IsEnabledClock</code>
	MIFEN	<code>LL_AHB1_GRP1_DisableClock</code>
		<code>LL_AHB1_GRP1_EnableClock</code>
		<code>LL_AHB1_GRP1_IsEnabledClock</code>
	RNGEN	<code>LL_AHB1_GRP1_DisableClock</code>

Register	Field	Function
AHBRSTR	TSCEN	<i>LL_AHB1_GRP1_EnableClock</i>
		<i>LL_AHB1_GRP1_IsEnabledClock</i>
	TSCEN	<i>LL_AHB1_GRP1_DisableClock</i>
		<i>LL_AHB1_GRP1_EnableClock</i>
		<i>LL_AHB1_GRP1_IsEnabledClock</i>
	CRCRST	<i>LL_AHB1_GRP1_ForceReset</i>
		<i>LL_AHB1_GRP1_ReleaseReset</i>
	CRYPRST	<i>LL_AHB1_GRP1_ForceReset</i>
		<i>LL_AHB1_GRP1_ReleaseReset</i>
	DMARST	<i>LL_AHB1_GRP1_ForceReset</i>
		<i>LL_AHB1_GRP1_ReleaseReset</i>
AHBSMENR	MIFRST	<i>LL_AHB1_GRP1_ForceReset</i>
		<i>LL_AHB1_GRP1_ReleaseReset</i>
	RNGRST	<i>LL_AHB1_GRP1_ForceReset</i>
		<i>LL_AHB1_GRP1_ReleaseReset</i>
	TSCRST	<i>LL_AHB1_GRP1_ForceReset</i>
		<i>LL_AHB1_GRP1_ReleaseReset</i>
	CRCSMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
	CRYPSMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
	DMASMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
	MIFSMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
	RNGSMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
	SRAMSMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
	TSCSMEN	<i>LL_AHB1_GRP1_DisableClockSleep</i>
		<i>LL_AHB1_GRP1_EnableClockSleep</i>
APB1ENR	CRSEN	<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
	Dacen	<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>

Register	Field	Function
I2C1EN		<i>LL_APB1_GRP1_IsEnabledClock</i>
		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
I2C2EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
I2C3EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
LCDEN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
LPTIM1EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
LPUART1EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
PWREN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
SPI2EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
TIM2EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
TIM3EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
TIM6EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
TIM7EN		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>

Register	Field	Function
APB1RSTR	USART2EN	<i>LL_APB1_GRP1_IsEnabledClock</i>
		<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
	USART4EN	<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
	USART5EN	<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
	USBEN	<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
	WWDGEN	<i>LL_APB1_GRP1_DisableClock</i>
		<i>LL_APB1_GRP1_EnableClock</i>
		<i>LL_APB1_GRP1_IsEnabledClock</i>
	CRSRST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	DACRST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	I2C1RST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	I2C2RST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	I2C3RST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	LCDRST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	LPTIM1RST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	LPUART1RST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	PWRRST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>
	SPI2RST	<i>LL_APB1_GRP1_ForceReset</i>
		<i>LL_APB1_GRP1_ReleaseReset</i>

Register	Field	Function
	TIM2RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	TIM3RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	TIM6RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	TIM7RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	USART2RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	USART4RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	USART5RST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	USBRST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
	WWDGRST	<code>LL_APB1_GRP1_ForceReset</code>
		<code>LL_APB1_GRP1_ReleaseReset</code>
APB1SMENR	CRSSMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	DACSMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	I2C1SMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	I2C2SMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	I2C3SMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	LCDSMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	LPTIM1SMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	LPUART1SMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>
	PWRSMEN	<code>LL_APB1_GRP1_DisableClockSleep</code>
		<code>LL_APB1_GRP1_EnableClockSleep</code>

Register	Field	Function
	SPI2SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	TIM2SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	TIM3SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	TIM6SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	TIM7SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	USART2SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	USART4SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	USART5SMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	USBSMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	WWDGSMEN	<i>LL_APB1_GRP1_DisableClockSleep</i>
		<i>LL_APB1_GRP1_EnableClockSleep</i>
	ADCEN	<i>LL_APB2_GRP1_DisableClock</i>
		<i>LL_APB2_GRP1_EnableClock</i>
		<i>LL_APB2_GRP1_IsEnabledClock</i>
	DBGEN	<i>LL_APB2_GRP1_DisableClock</i>
		<i>LL_APB2_GRP1_EnableClock</i>
		<i>LL_APB2_GRP1_IsEnabledClock</i>
	FWEN	<i>LL_APB2_GRP1_DisableClock</i>
		<i>LL_APB2_GRP1_EnableClock</i>
		<i>LL_APB2_GRP1_IsEnabledClock</i>
	SPI1EN	<i>LL_APB2_GRP1_DisableClock</i>
		<i>LL_APB2_GRP1_EnableClock</i>
		<i>LL_APB2_GRP1_IsEnabledClock</i>
	SYSCFGEN	<i>LL_APB2_GRP1_DisableClock</i>
		<i>LL_APB2_GRP1_EnableClock</i>
		<i>LL_APB2_GRP1_IsEnabledClock</i>
	TIM21EN	<i>LL_APB2_GRP1_DisableClock</i>

Register	Field	Function
APB2RSTR	TIM22EN	<code>LL_APB2_GRP1_EnableClock</code>
		<code>LL_APB2_GRP1_IsEnabledClock</code>
	USART1EN	<code>LL_APB2_GRP1_DisableClock</code>
		<code>LL_APB2_GRP1_EnableClock</code>
	USART1EN	<code>LL_APB2_GRP1_IsEnabledClock</code>
	ADCRST	<code>LL_APB2_GRP1_DisableClock</code>
		<code>LL_APB2_GRP1_EnableClock</code>
		<code>LL_APB2_GRP1_IsEnabledClock</code>
	DBGRST	<code>LL_APB2_GRP1_ForceReset</code>
		<code>LL_APB2_GRP1_ReleaseReset</code>
	SPI1RST	<code>LL_APB2_GRP1_ForceReset</code>
		<code>LL_APB2_GRP1_ReleaseReset</code>
	SYSCFGRST	<code>LL_APB2_GRP1_ForceReset</code>
		<code>LL_APB2_GRP1_ReleaseReset</code>
	TIM21RST	<code>LL_APB2_GRP1_ForceReset</code>
		<code>LL_APB2_GRP1_ReleaseReset</code>
	TIM22RST	<code>LL_APB2_GRP1_ForceReset</code>
		<code>LL_APB2_GRP1_ReleaseReset</code>
	USART1RST	<code>LL_APB2_GRP1_ForceReset</code>
		<code>LL_APB2_GRP1_ReleaseReset</code>
APB2SMENR	ADCSMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>
	DBGSMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>
	SPI1SMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>
	SYSCFGSMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>
	TIM21SMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>
	TIM22SMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>
	USART1SMEN	<code>LL_APB2_GRP1_DisableClockSleep</code>
		<code>LL_APB2_GRP1_EnableClockSleep</code>

Register	Field	Function
IOPENR	GPIOAEN	<a href="#"><i>LL_IOP_GRP1_DisableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_IsEnabledClock</i></a>
	GPIOBEN	<a href="#"><i>LL_IOP_GRP1_DisableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_IsEnabledClock</i></a>
	GPIOCEN	<a href="#"><i>LL_IOP_GRP1_DisableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_IsEnabledClock</i></a>
	GPIODEN	<a href="#"><i>LL_IOP_GRP1_DisableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_IsEnabledClock</i></a>
IOPRSTR	GPIOEEN	<a href="#"><i>LL_IOP_GRP1_DisableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_IsEnabledClock</i></a>
	GPIOHEN	<a href="#"><i>LL_IOP_GRP1_DisableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClock</i></a>
		<a href="#"><i>LL_IOP_GRP1_IsEnabledClock</i></a>
	GPIOASMEN	<a href="#"><i>LL_IOP_GRP1_ForceReset</i></a>
		<a href="#"><i>LL_IOP_GRP1_ReleaseReset</i></a>
	GPIOBSMEN	<a href="#"><i>LL_IOP_GRP1_ForceReset</i></a>
		<a href="#"><i>LL_IOP_GRP1_ReleaseReset</i></a>
	GPIOCSMEN	<a href="#"><i>LL_IOP_GRP1_ForceReset</i></a>
		<a href="#"><i>LL_IOP_GRP1_ReleaseReset</i></a>
	GPIODSMEN	<a href="#"><i>LL_IOP_GRP1_ForceReset</i></a>
		<a href="#"><i>LL_IOP_GRP1_ReleaseReset</i></a>
	GPIOESMEN	<a href="#"><i>LL_IOP_GRP1_ForceReset</i></a>
		<a href="#"><i>LL_IOP_GRP1_ReleaseReset</i></a>
	GPIOHSMEN	<a href="#"><i>LL_IOP_GRP1_ForceReset</i></a>
		<a href="#"><i>LL_IOP_GRP1_ReleaseReset</i></a>
IOPSMENR	GPIOARST	<a href="#"><i>LL_IOP_GRP1_DisableClockSleep</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClockSleep</i></a>
	GPIOBRST	<a href="#"><i>LL_IOP_GRP1_DisableClockSleep</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClockSleep</i></a>
	GPIOCRST	<a href="#"><i>LL_IOP_GRP1_DisableClockSleep</i></a>
		<a href="#"><i>LL_IOP_GRP1_EnableClockSleep</i></a>

Register	Field	Function
	GPIODRST	<i>LL_IOP_GRP1_DisableClockSleep</i>
		<i>LL_IOP_GRP1_EnableClockSleep</i>
	GPIOERST	<i>LL_IOP_GRP1_DisableClockSleep</i>
		<i>LL_IOP_GRP1_EnableClockSleep</i>
	GPIOHRST	<i>LL_IOP_GRP1_DisableClockSleep</i>
		<i>LL_IOP_GRP1_EnableClockSleep</i>

## 78.3 COMP

Table 27: Correspondence between COMP registers and COMP low-layer driver functions

Register	Field	Function
COMP1_CSR	COMP1EN	<i>LL_COMP_Disable</i>
		<i>LL_COMP_Enable</i>
		<i>LL_COMP_IsEnabled</i>
	COMP1INNSEL	<i>LL_COMP_GetInputMinus</i>
		<i>LL_COMP_SetInputMinus</i>
	COMP1LOCK	<i>LL_COMP_IsLocked</i>
		<i>LL_COMP_Lock</i>
	COMP1LPTIMIN1	<i>LL_COMP_GetOutputLPTIM</i>
		<i>LL_COMP_SetOutputLPTIM</i>
	COMP1POLARITY	<i>LL_COMP_GetOutputPolarity</i>
		<i>LL_COMP_SetOutputPolarity</i>
	COMP1VALUE	<i>LL_COMP_ReadOutputLevel</i>
COMP2_CSR	COMP2EN	<i>LL_COMP_GetCommonWindowMode</i>
		<i>LL_COMP_SetCommonWindowMode</i>
		<i>LL_COMP_Disable</i>
		<i>LL_COMP_Enable</i>
		<i>LL_COMP_IsEnabled</i>
	COMP2INNSEL	<i>LL_COMP_ConfigInputs</i>
		<i>LL_COMP_GetInputMinus</i>
		<i>LL_COMP_SetInputMinus</i>
	COMP2INPSEL	<i>LL_COMP_ConfigInputs</i>
		<i>LL_COMP_GetInputPlus</i>
		<i>LL_COMP_SetInputPlus</i>
COMP2LOCK	COMP2LOCK	<i>LL_COMP_IsLocked</i>
		<i>LL_COMP_Lock</i>
	COMP2LPTIMIN1	<i>LL_COMP_GetOutputLPTIM</i>

Register	Field	Function
	COMP2LPTIMIN2	<i>LL_COMP_SetOutputLPTIM</i>
		<i>LL_COMP_GetOutputLPTIM</i>
		<i>LL_COMP_SetOutputLPTIM</i>
	COMP2POLARITY	<i>LL_COMP_GetOutputPolarity</i>
		<i>LL_COMP_SetOutputPolarity</i>
	COMP2SPEED	<i>LL_COMP_GetPowerMode</i>
		<i>LL_COMP_SetPowerMode</i>
	COMP2VALUE	<i>LL_COMP_ReadOutputLevel</i>

## 78.4 CORTEX

Table 28: Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	<i>LL_MPU_Disable</i>
		<i>LL_MPU_Enable</i>
		<i>LL_MPU_IsEnabled</i>
MPU_RASR	A	<i>LL_MPU_ConfigRegion</i>
	B	<i>LL_MPU_ConfigRegion</i>
	C	<i>LL_MPU_ConfigRegion</i>
	ENABLE	<i>LL_MPU_DisableRegion</i>
		<i>LL_MPU_EnableRegion</i>
	S	<i>LL_MPU_ConfigRegion</i>
	SIZE	<i>LL_MPU_ConfigRegion</i>
MPU_RBAR	XN	<i>LL_MPU_ConfigRegion</i>
	ADDR	<i>LL_MPU_ConfigRegion</i>
MPU_RNR	REGION	<i>LL_MPU_ConfigRegion</i>
	REGION	<i>LL_MPU_DisableRegion</i>
SCB_CPUID	ARCHITECTURE	<i>LL_CPUID_GetArchitecture</i>
	IMPLEMENTER	<i>LL_CPUID_GetImplementer</i>
	PARTNO	<i>LL_CPUID_GetParNo</i>
	REVISION	<i>LL_CPUID_GetRevision</i>
	VARIANT	<i>LL_CPUID_GetVariant</i>
SCB_SCR	SEVEONPEND	<i>LL_LPM_DisableEventOnPend</i>
		<i>LL_LPM_EnableEventOnPend</i>
	SLEEPDEEP	<i>LL_LPM_EnableDeepSleep</i>
		<i>LL_LPM_EnableSleep</i>

Register	Field	Function
STK_CTRL	SLEEPONEXIT	<a href="#"><i>LL_LPM_DisableSleepOnExit</i></a>
		<a href="#"><i>LL_LPM_EnableSleepOnExit</i></a>
	CLKSOURCE	<a href="#"><i>LL_SYSTICK_GetClkSource</i></a>
		<a href="#"><i>LL_SYSTICK_SetClkSource</i></a>
	COUNTFLAG	<a href="#"><i>LL_SYSTICK_IsActiveCounterFlag</i></a>
	TICKINT	<a href="#"><i>LL_SYSTICK_DisableIT</i></a>
		<a href="#"><i>LL_SYSTICK_EnableIT</i></a>
		<a href="#"><i>LL_SYSTICK_IsEnabledIT</i></a>

## 78.5 CRC

Table 29: Correspondence between CRC registers and CRC low-layer driver functions

Register	Field	Function
CR	POLYSIZE	<a href="#"><i>LL_CRC_GetPolynomialSize</i></a>
		<a href="#"><i>LL_CRC_SetPolynomialSize</i></a>
	RESET	<a href="#"><i>LL_CRC_ResetCRCCalculationUnit</i></a>
	REV_IN	<a href="#"><i>LL_CRC_GetInputDataReverseMode</i></a>
		<a href="#"><i>LL_CRC_SetInputDataReverseMode</i></a>
	REV_OUT	<a href="#"><i>LL_CRC_GetOutputDataReverseMode</i></a>
		<a href="#"><i>LL_CRC_SetOutputDataReverseMode</i></a>
DR	DR	<a href="#"><i>LL_CRC_FeedData16</i></a>
		<a href="#"><i>LL_CRC_FeedData32</i></a>
		<a href="#"><i>LL_CRC_FeedData8</i></a>
		<a href="#"><i>LL_CRC_ReadData16</i></a>
		<a href="#"><i>LL_CRC_ReadData32</i></a>
		<a href="#"><i>LL_CRC_ReadData7</i></a>
		<a href="#"><i>LL_CRC_ReadData8</i></a>
IDR	IDR	<a href="#"><i>LL_CRC_Read_IDR</i></a>
		<a href="#"><i>LL_CRC_Write_IDR</i></a>
INIT	INIT	<a href="#"><i>LL_CRC_GetInitialData</i></a>
		<a href="#"><i>LL_CRC_SetInitialData</i></a>
POL	POL	<a href="#"><i>LL_CRC_GetPolynomialCoef</i></a>
		<a href="#"><i>LL_CRC_SetPolynomialCoef</i></a>

## 78.6 CRS

Table 30: Correspondence between CRS registers and CRS low-layer driver functions

Register	Field	Function
CFGR	FELIM	<i>LL_CRS_ConfigSynchronization</i>
		<i>LL_CRS_GetFreqErrorLimit</i>
		<i>LL_CRS_SetFreqErrorLimit</i>
	RELOAD	<i>LL_CRS_ConfigSynchronization</i>
		<i>LL_CRS_GetReloadCounter</i>
		<i>LL_CRS_SetReloadCounter</i>
	SYNCDIV	<i>LL_CRS_ConfigSynchronization</i>
		<i>LL_CRS_GetSyncDivider</i>
		<i>LL_CRS_SetSyncDivider</i>
	SYNCPOL	<i>LL_CRS_ConfigSynchronization</i>
		<i>LL_CRS_GetSyncPolarity</i>
		<i>LL_CRS_SetSyncPolarity</i>
	SYNCSRC	<i>LL_CRS_ConfigSynchronization</i>
		<i>LL_CRS_GetSyncSignalSource</i>
		<i>LL_CRS_SetSyncSignalSource</i>
CR	AUTOTRIMEN	<i>LL_CRS_DisableAutoTrimming</i>
		<i>LL_CRS_EnableAutoTrimming</i>
		<i>LL_CRS_IsEnabledAutoTrimming</i>
	CEN	<i>LL_CRS_DisableFreqErrorCounter</i>
		<i>LL_CRS_EnableFreqErrorCounter</i>
		<i>LL_CRS_IsEnabledFreqErrorCounter</i>
	ERRIE	<i>LL_CRS_DisableIT_ERR</i>
		<i>LL_CRS_EnableIT_ERR</i>
		<i>LL_CRS_IsEnabledIT_ERR</i>
	EYNCIE	<i>LL_CRS_DisableIT_ESYNC</i>
		<i>LL_CRS_EnableIT_ESYNC</i>
		<i>LL_CRS_IsEnabledIT_ESYNC</i>
	SWSYNC	<i>LL_CRS_GenerateEvent_SWSYNC</i>
	SYNCOKIE	<i>LL_CRS_DisableIT_SYNCOK</i>
		<i>LL_CRS_EnableIT_SYNCOK</i>
		<i>LL_CRS_IsEnabledIT_SYNCOK</i>
	SYNCWARNIE	<i>LL_CRS_DisableIT_SYNCWARN</i>
		<i>LL_CRS_EnableIT_SYNCWARN</i>
		<i>LL_CRS_IsEnabledIT_SYNCWARN</i>

Register	Field	Function
	TRIM	<i>LL_CRS_ConfigSynchronization</i>
		<i>LL_CRS_GetHSI48SmoothTrimming</i>
		<i>LL_CRS_SetHSI48SmoothTrimming</i>
ICR	ERRC	<i>LL_CRS_ClearFlag_ERR</i>
	ESYNCC	<i>LL_CRS_ClearFlag_ESYNC</i>
	SYNCOKC	<i>LL_CRS_ClearFlag_SYNCOK</i>
	SYNCWARNC	<i>LL_CRS_ClearFlag_SYNCWARN</i>
ISR	ERRF	<i>LL_CRS_IsActiveFlag_ERR</i>
	ESYNCF	<i>LL_CRS_IsActiveFlag_ESYNC</i>
	FECAP	<i>LL_CRS_GetFreqErrorCapture</i>
	FEDIR	<i>LL_CRS_GetFreqErrorDirection</i>
	SYNCERR	<i>LL_CRS_IsActiveFlag_SYNCERR</i>
	SYNCMISS	<i>LL_CRS_IsActiveFlag_SYNCMISS</i>
	SYNCOKF	<i>LL_CRS_IsActiveFlag_SYNCOK</i>
	SYNCWARNF	<i>LL_CRS_IsActiveFlag_SYNCWARN</i>
	TRIMOVF	<i>LL_CRS_IsActiveFlag_TRIMOVF</i>

## 78.7 DAC

Table 31: Correspondence between DAC registers and DAC low-layer driver functions

Register	Field	Function
CR	BOFF1	<i>LL_DAC_GetOutputBuffer</i>
		<i>LL_DAC_SetOutputBuffer</i>
	BOFF2	<i>LL_DAC_GetOutputBuffer</i>
		<i>LL_DAC_SetOutputBuffer</i>
	DMAEN1	<i>LL_DAC_DisableDMAReq</i>
		<i>LL_DAC_EnableDMAReq</i>
		<i>LL_DAC_IsDMAReqEnabled</i>
	DMAEN2	<i>LL_DAC_DisableDMAReq</i>
		<i>LL_DAC_EnableDMAReq</i>
		<i>LL_DAC_IsDMAReqEnabled</i>
	DMAUDRIE1	<i>LL_DAC_DisableIT_DMAUDR1</i>
		<i>LL_DAC_EnableIT_DMAUDR1</i>
		<i>LL_DAC_IsEnabledIT_DMAUDR1</i>
	DMAUDRIE2	<i>LL_DAC_DisableIT_DMAUDR2</i>
		<i>LL_DAC_EnableIT_DMAUDR2</i>
		<i>LL_DAC_IsEnabledIT_DMAUDR2</i>

Register	Field	Function
	EN1	<i>LL_DAC_Disable</i>
		<i>LL_DAC_Enable</i>
		<i>LL_DAC_IsEnabled</i>
	EN2	<i>LL_DAC_Disable</i>
		<i>LL_DAC_Enable</i>
		<i>LL_DAC_IsEnabled</i>
	MAMP1	<i>LL_DAC_GetWaveNoiseLFSR</i>
		<i>LL_DAC_GetWaveTriangleAmplitude</i>
		<i>LL_DAC_SetWaveNoiseLFSR</i>
		<i>LL_DAC_SetWaveTriangleAmplitude</i>
	MAMP2	<i>LL_DAC_GetWaveNoiseLFSR</i>
		<i>LL_DAC_GetWaveTriangleAmplitude</i>
		<i>LL_DAC_SetWaveNoiseLFSR</i>
		<i>LL_DAC_SetWaveTriangleAmplitude</i>
	TEN1	<i>LL_DAC_DisableTrigger</i>
		<i>LL_DAC_EnableTrigger</i>
		<i>LL_DAC_IsTriggerEnabled</i>
	TEN2	<i>LL_DAC_DisableTrigger</i>
		<i>LL_DAC_EnableTrigger</i>
		<i>LL_DAC_IsTriggerEnabled</i>
	TSEL1	<i>LL_DAC_GetTriggerSource</i>
		<i>LL_DAC_SetTriggerSource</i>
	TSEL2	<i>LL_DAC_GetTriggerSource</i>
		<i>LL_DAC_SetTriggerSource</i>
	WAVE1	<i>LL_DAC_GetWaveAutoGeneration</i>
		<i>LL_DAC_SetWaveAutoGeneration</i>
	WAVE2	<i>LL_DAC_GetWaveAutoGeneration</i>
		<i>LL_DAC_SetWaveAutoGeneration</i>
DHR12L1	DACC1DHR	<i>LL_DAC_ConvertData12LeftAligned</i>
		<i>LL_DAC_DMA_GetRegAddr</i>
DHR12L2	DACC2DHR	<i>LL_DAC_ConvertData12LeftAligned</i>
		<i>LL_DAC_DMA_GetRegAddr</i>
DHR12LD	DACC1DHR	<i>LL_DAC_ConvertDualData12LeftAligned</i>
	DACC2DHR	<i>LL_DAC_ConvertDualData12LeftAligned</i>
DHR12R1	DACC1DHR	<i>LL_DAC_ConvertData12RightAligned</i>
		<i>LL_DAC_DMA_GetRegAddr</i>

Register	Field	Function
DHR12R2	DACC2DHR	<i>LL_DAC_ConvertData12RightAligned</i>
		<i>LL_DAC_DMA_GetRegAddr</i>
DHR12RD	DACC1DHR	<i>LL_DAC_ConvertDualData12RightAligned</i>
	DACC2DHR	<i>LL_DAC_ConvertDualData12RightAligned</i>
DHR8R1	DACC1DHR	<i>LL_DAC_ConvertData8RightAligned</i>
		<i>LL_DAC_DMA_GetRegAddr</i>
DHR8R2	DACC2DHR	<i>LL_DAC_ConvertData8RightAligned</i>
		<i>LL_DAC_DMA_GetRegAddr</i>
DHR8RD	DACC1DHR	<i>LL_DAC_ConvertDualData8RightAligned</i>
	DACC2DHR	<i>LL_DAC_ConvertDualData8RightAligned</i>
DOR1	DACC1DOR	<i>LL_DAC_RetrieveOutputData</i>
DOR2	DACC2DOR	<i>LL_DAC_RetrieveOutputData</i>
SR	DMAUDR1	<i>LL_DAC_ClearFlag_DMAUDR1</i>
		<i>LL_DAC_IsActiveFlag_DMAUDR1</i>
	DMAUDR2	<i>LL_DAC_ClearFlag_DMAUDR2</i>
		<i>LL_DAC_IsActiveFlag_DMAUDR2</i>
SWTRIGR	SWTRIG1	<i>LL_DAC_TrigSWConversion</i>
	SWTRIG2	<i>LL_DAC_TrigSWConversion</i>

## 78.8 DMA

Table 32: Correspondence between DMA registers and DMA low-layer driver functions

Register	Field	Function
CCR	CIRC	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetMode</i>
		<i>LL_DMA_SetMode</i>
	DIR	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetDataTransferDirection</i>
		<i>LL_DMA_SetDataTransferDirection</i>
	EN	<i>LL_DMA_DisableChannel</i>
		<i>LL_DMA_EnableChannel</i>
		<i>LL_DMA_IsEnabledChannel</i>
	HTIE	<i>LL_DMA_DisableIT_HT</i>
		<i>LL_DMA_EnableIT_HT</i>
		<i>LL_DMA_IsEnabledIT_HT</i>
	MEM2MEM	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetDataTransferDirection</i>

Register	Field	Function
	MINC	<i>LL_DMA_SetDataTransferDirection</i>
		<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetMemoryIncMode</i>
		<i>LL_DMA_SetMemoryIncMode</i>
	MSIZE	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetMemorySize</i>
		<i>LL_DMA_SetMemorySize</i>
	PINC	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetPeriphIncMode</i>
		<i>LL_DMA_SetPeriphIncMode</i>
	PL	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetChannelPriorityLevel</i>
		<i>LL_DMA_SetChannelPriorityLevel</i>
	PSIZE	<i>LL_DMA_ConfigTransfer</i>
		<i>LL_DMA_GetPeriphSize</i>
		<i>LL_DMA_SetPeriphSize</i>
	TCIE	<i>LL_DMA_DisableIT_TC</i>
		<i>LL_DMA_EnableIT_TC</i>
		<i>LL_DMA_IsEnabledIT_TC</i>
	TEIE	<i>LL_DMA_DisableIT_TE</i>
		<i>LL_DMA_EnableIT_TE</i>
		<i>LL_DMA_IsEnabledIT_TE</i>
CMAR	MA	<i>LL_DMA_ConfigAddresses</i>
		<i>LL_DMA_GetM2MDstAddress</i>
		<i>LL_DMA_GetMemoryAddress</i>
		<i>LL_DMA_SetM2MDstAddress</i>
		<i>LL_DMA_SetMemoryAddress</i>
CNDTR	NDT	<i>LL_DMA_GetDataLength</i>
		<i>LL_DMA_SetDataLength</i>
CPAR	PA	<i>LL_DMA_ConfigAddresses</i>
		<i>LL_DMA_GetM2MSrcAddress</i>
		<i>LL_DMA_GetPeriphAddress</i>
		<i>LL_DMA_SetM2MSrcAddress</i>
		<i>LL_DMA_SetPeriphAddress</i>
CSELR	C1S	<i>LL_DMA_GetPeriphRequest</i>
		<i>LL_DMA_SetPeriphRequest</i>

Register	Field	Function
IFCR	C2S	<a href="#"><i>LL_DMA_GetPeriphRequest</i></a>
		<a href="#"><i>LL_DMA_SetPeriphRequest</i></a>
	C3S	<a href="#"><i>LL_DMA_GetPeriphRequest</i></a>
		<a href="#"><i>LL_DMA_SetPeriphRequest</i></a>
	C4S	<a href="#"><i>LL_DMA_GetPeriphRequest</i></a>
		<a href="#"><i>LL_DMA_SetPeriphRequest</i></a>
	C5S	<a href="#"><i>LL_DMA_GetPeriphRequest</i></a>
		<a href="#"><i>LL_DMA_SetPeriphRequest</i></a>
	C6S	<a href="#"><i>LL_DMA_GetPeriphRequest</i></a>
		<a href="#"><i>LL_DMA_SetPeriphRequest</i></a>
	C7S	<a href="#"><i>LL_DMA_GetPeriphRequest</i></a>
		<a href="#"><i>LL_DMA_SetPeriphRequest</i></a>
	CGIF1	<a href="#"><i>LL_DMA_ClearFlag_GI1</i></a>
	CGIF2	<a href="#"><i>LL_DMA_ClearFlag_GI2</i></a>
	CGIF3	<a href="#"><i>LL_DMA_ClearFlag_GI3</i></a>
	CGIF4	<a href="#"><i>LL_DMA_ClearFlag_GI4</i></a>
	CGIF5	<a href="#"><i>LL_DMA_ClearFlag_GI5</i></a>
	CGIF6	<a href="#"><i>LL_DMA_ClearFlag_GI6</i></a>
	CGIF7	<a href="#"><i>LL_DMA_ClearFlag_GI7</i></a>
	CHTIF1	<a href="#"><i>LL_DMA_ClearFlag_HT1</i></a>
	CHTIF2	<a href="#"><i>LL_DMA_ClearFlag_HT2</i></a>
	CHTIF3	<a href="#"><i>LL_DMA_ClearFlag_HT3</i></a>
	CHTIF4	<a href="#"><i>LL_DMA_ClearFlag_HT4</i></a>
	CHTIF5	<a href="#"><i>LL_DMA_ClearFlag_HT5</i></a>
	CHTIF6	<a href="#"><i>LL_DMA_ClearFlag_HT6</i></a>
	CHTIF7	<a href="#"><i>LL_DMA_ClearFlag_HT7</i></a>
	CTCIF1	<a href="#"><i>LL_DMA_ClearFlag_TC1</i></a>
	CTCIF2	<a href="#"><i>LL_DMA_ClearFlag_TC2</i></a>
	CTCIF3	<a href="#"><i>LL_DMA_ClearFlag_TC3</i></a>
	CTCIF4	<a href="#"><i>LL_DMA_ClearFlag_TC4</i></a>
	CTCIF5	<a href="#"><i>LL_DMA_ClearFlag_TC5</i></a>
	CTCIF6	<a href="#"><i>LL_DMA_ClearFlag_TC6</i></a>
	CTCIF7	<a href="#"><i>LL_DMA_ClearFlag_TC7</i></a>
	CTEIF1	<a href="#"><i>LL_DMA_ClearFlag_TE1</i></a>
	CTEIF2	<a href="#"><i>LL_DMA_ClearFlag_TE2</i></a>
	CTEIF3	<a href="#"><i>LL_DMA_ClearFlag_TE3</i></a>

Register	Field	Function
ISR	CTEIF4	<i>LL_DMA_ClearFlag_TE4</i>
	CTEIF5	<i>LL_DMA_ClearFlag_TE5</i>
	CTEIF6	<i>LL_DMA_ClearFlag_TE6</i>
	CTEIF7	<i>LL_DMA_ClearFlag_TE7</i>
	GIF1	<i>LL_DMA_IsActiveFlag_GI1</i>
	GIF2	<i>LL_DMA_IsActiveFlag_GI2</i>
	GIF3	<i>LL_DMA_IsActiveFlag_GI3</i>
	GIF4	<i>LL_DMA_IsActiveFlag_GI4</i>
	GIF5	<i>LL_DMA_IsActiveFlag_GI5</i>
	GIF6	<i>LL_DMA_IsActiveFlag_GI6</i>
	GIF7	<i>LL_DMA_IsActiveFlag_GI7</i>
	HTIF1	<i>LL_DMA_IsActiveFlag_HT1</i>
	HTIF2	<i>LL_DMA_IsActiveFlag_HT2</i>
	HTIF3	<i>LL_DMA_IsActiveFlag_HT3</i>
	HTIF4	<i>LL_DMA_IsActiveFlag_HT4</i>
	HTIF5	<i>LL_DMA_IsActiveFlag_HT5</i>
	HTIF6	<i>LL_DMA_IsActiveFlag_HT6</i>
	HTIF7	<i>LL_DMA_IsActiveFlag_HT7</i>
	TCIF1	<i>LL_DMA_IsActiveFlag_TC1</i>
	TCIF2	<i>LL_DMA_IsActiveFlag_TC2</i>
	TCIF3	<i>LL_DMA_IsActiveFlag_TC3</i>
	TCIF4	<i>LL_DMA_IsActiveFlag_TC4</i>
	TCIF5	<i>LL_DMA_IsActiveFlag_TC5</i>
	TCIF6	<i>LL_DMA_IsActiveFlag_TC6</i>
	TCIF7	<i>LL_DMA_IsActiveFlag_TC7</i>
	TEIF1	<i>LL_DMA_IsActiveFlag_TE1</i>
	TEIF2	<i>LL_DMA_IsActiveFlag_TE2</i>
	TEIF3	<i>LL_DMA_IsActiveFlag_TE3</i>
	TEIF4	<i>LL_DMA_IsActiveFlag_TE4</i>
	TEIF5	<i>LL_DMA_IsActiveFlag_TE5</i>
	TEIF6	<i>LL_DMA_IsActiveFlag_TE6</i>
	TEIF7	<i>LL_DMA_IsActiveFlag_TE7</i>

## 78.9 EXTI

Table 33: Correspondence between EXTI registers and EXTI low-layer driver functions

Register	Field	Function
EMR	EMx	<i>LL_EXTI_DisableEvent_0_31</i>
		<i>LL_EXTI_EnableEvent_0_31</i>
		<i>LL_EXTI_IsEnabledEvent_0_31</i>
FTSR	FTx	<i>LL_EXTI_DisableFallingTrig_0_31</i>
		<i>LL_EXTI_EnableFallingTrig_0_31</i>
		<i>LL_EXTI_IsEnabledFallingTrig_0_31</i>
IMR	IMx	<i>LL_EXTI_DisableIT_0_31</i>
		<i>LL_EXTI_EnableIT_0_31</i>
		<i>LL_EXTI_IsEnabledIT_0_31</i>
PR	PIFx	<i>LL_EXTI_ClearFlag_0_31</i>
		<i>LL_EXTI_IsActiveFlag_0_31</i>
		<i>LL_EXTI_ReadFlag_0_31</i>
RTSR	RTx	<i>LL_EXTI_DisableRisingTrig_0_31</i>
		<i>LL_EXTI_EnableRisingTrig_0_31</i>
		<i>LL_EXTI_IsEnabledRisingTrig_0_31</i>
SWIER	SWIx	<i>LL_EXTI_GenerateSWI_0_31</i>

## 78.10 GPIO

Table 34: Correspondence between GPIO registers and GPIO low-layer driver functions

Register	Field	Function
AFRH	AFSELy	<i>LL_GPIO_GetAFPin_8_15</i>
		<i>LL_GPIO_SetAFPin_8_15</i>
AFRL	AFSELy	<i>LL_GPIO_GetAFPin_0_7</i>
		<i>LL_GPIO_SetAFPin_0_7</i>
BRR	BRy	<i>LL_GPIO_ResetOutputPin</i>
BSRR	BSy	<i>LL_GPIO_SetOutputPin</i>
IDR	IDy	<i>LL_GPIO_IsInputPinSet</i>
		<i>LL_GPIO_ReadInputPort</i>
LCKR	LCKK	<i>LL_GPIO_IsAnyPinLocked</i>
		<i>LL_GPIO_LockPin</i>
LCKY	LCKy	<i>LL_GPIO_IsPinLocked</i>
MODER	MODEy	<i>LL_GPIO_GetPinMode</i>
		<i>LL_GPIO_SetPinMode</i>
ODR	ODy	<i>LL_GPIO_IsOutputPinSet</i>

Register	Field	Function
		<i>LL_GPIO_ReadOutputPort</i>
		<i>LL_GPIO_TogglePin</i>
		<i>LL_GPIO_WriteOutputPort</i>
OSPEEDR	OSPEEDy	<i>LL_GPIO_GetPinSpeed</i>
		<i>LL_GPIO_SetPinSpeed</i>
OTYPER	OTy	<i>LL_GPIO_GetPinOutputType</i>
		<i>LL_GPIO_SetPinOutputType</i>
PUPDR	PUPDy	<i>LL_GPIO_GetPinPull</i>
		<i>LL_GPIO_SetPinPull</i>

## 78.11 I2C

Table 35: Correspondence between I2C registers and I2C low-layer driver functions

Register	Field	Function
CR1	ADDRIE	<i>LL_I2C_DisableIT_ADDR</i>
		<i>LL_I2C_EnableIT_ADDR</i>
		<i>LL_I2C_IsEnabledIT_ADDR</i>
	ALERTEN	<i>LL_I2C_DisableSMBusAlert</i>
		<i>LL_I2C_EnableSMBusAlert</i>
		<i>LL_I2C_IsEnabledSMBusAlert</i>
	ANFOFF	<i>LL_I2C_ConfigFilters</i>
		<i>LL_I2C_DisableAnalogFilter</i>
		<i>LL_I2C_EnableAnalogFilter</i>
		<i>LL_I2C_IsEnabledAnalogFilter</i>
	DNF	<i>LL_I2C_ConfigFilters</i>
		<i>LL_I2C_GetDigitalFilter</i>
		<i>LL_I2C_SetDigitalFilter</i>
	ERRIE	<i>LL_I2C_DisableIT_ERR</i>
		<i>LL_I2C_EnableIT_ERR</i>
		<i>LL_I2C_IsEnabledIT_ERR</i>
	GCEN	<i>LL_I2C_DisableGeneralCall</i>
		<i>LL_I2C_EnableGeneralCall</i>
		<i>LL_I2C_IsEnabledGeneralCall</i>
	NACKIE	<i>LL_I2C_DisableIT_NACK</i>
		<i>LL_I2C_EnableIT_NACK</i>
		<i>LL_I2C_IsEnabledIT_NACK</i>
	NOSTRETCH	<i>LL_I2C_DisableClockStretching</i>

Register	Field	Function
		<i>LL_I2C_EnableClockStretching</i>
		<i>LL_I2C_IsEnabledClockStretching</i>
	PE	<i>LL_I2C_Disable</i>
		<i>LL_I2C_Enable</i>
		<i>LL_I2C_IsEnabled</i>
	PECEN	<i>LL_I2C_DisableSMBusPEC</i>
		<i>LL_I2C_EnableSMBusPEC</i>
		<i>LL_I2C_IsEnabledSMBusPEC</i>
	RXDMAEN	<i>LL_I2C_DisableDMAReq_RX</i>
		<i>LL_I2C_EnableDMAReq_RX</i>
		<i>LL_I2C_IsEnabledDMAReq_RX</i>
	RXIE	<i>LL_I2C_DisableIT_RX</i>
		<i>LL_I2C_EnableIT_RX</i>
		<i>LL_I2C_IsEnabledIT_RX</i>
	SBC	<i>LL_I2C_DisableSlaveByteControl</i>
		<i>LL_I2C_EnableSlaveByteControl</i>
		<i>LL_I2C_IsEnabledSlaveByteControl</i>
	SMBDEN	<i>LL_I2C_GetMode</i>
		<i>LL_I2C_SetMode</i>
	SMBHEN	<i>LL_I2C_GetMode</i>
		<i>LL_I2C_SetMode</i>
	STOPIE	<i>LL_I2C_DisableIT_STOP</i>
		<i>LL_I2C_EnableIT_STOP</i>
		<i>LL_I2C_IsEnabledIT_STOP</i>
	TCIE	<i>LL_I2C_DisableIT_TC</i>
		<i>LL_I2C_EnableIT_TC</i>
		<i>LL_I2C_IsEnabledIT_TC</i>
	TXDMAEN	<i>LL_I2C_DisableDMAReq_TX</i>
		<i>LL_I2C_EnableDMAReq_TX</i>
		<i>LL_I2C_IsEnabledDMAReq_TX</i>
	TXIE	<i>LL_I2C_DisableIT_TX</i>
		<i>LL_I2C_EnableIT_TX</i>
		<i>LL_I2C_IsEnabledIT_TX</i>
	WUPEN	<i>LL_I2C_DisableWakeUpFromStop</i>
		<i>LL_I2C_EnableWakeUpFromStop</i>
		<i>LL_I2C_IsEnabledWakeUpFromStop</i>

Register	Field	Function
CR2	ADD10	<a href="#"><i>LL_I2C_GetMasterAddressingMode</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_SetMasterAddressingMode</i></a>
	AUTOEND	<a href="#"><i>LL_I2C_DisableAutoEndMode</i></a>
		<a href="#"><i>LL_I2C_EnableAutoEndMode</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_IsEnabledAutoEndMode</i></a>
	HEAD10R	<a href="#"><i>LL_I2C_DisableAuto10BitRead</i></a>
		<a href="#"><i>LL_I2C_EnableAuto10BitRead</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_IsEnabledAuto10BitRead</i></a>
	NACK	<a href="#"><i>LL_I2C_AcknowledgeNextData</i></a>
	NBYTES	<a href="#"><i>LL_I2C_GetTransferSize</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_SetTransferSize</i></a>
	PECBYTE	<a href="#"><i>LL_I2C_EnableSMBusPECCCompare</i></a>
		<a href="#"><i>LL_I2C_IsEnabledSMBusPECCCompare</i></a>
	RD_WRN	<a href="#"><i>LL_I2C_GetTransferRequest</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_SetTransferRequest</i></a>
	RELOAD	<a href="#"><i>LL_I2C_DisableReloadMode</i></a>
		<a href="#"><i>LL_I2C_EnableReloadMode</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_IsEnabledReloadMode</i></a>
	SADD	<a href="#"><i>LL_I2C_GetSlaveAddr</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
		<a href="#"><i>LL_I2C_SetSlaveAddr</i></a>
	START	<a href="#"><i>LL_I2C_GenerateStartCondition</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
	STOP	<a href="#"><i>LL_I2C_GenerateStopCondition</i></a>
		<a href="#"><i>LL_I2C_HandleTransfer</i></a>
ICR	ADDRCF	<a href="#"><i>LL_I2C_ClearFlag_ADDR</i></a>
	ALERTCF	<a href="#"><i>LL_I2C_ClearSMBusFlag_ALERT</i></a>
	ARLOCF	<a href="#"><i>LL_I2C_ClearFlag_ARLO</i></a>
	BERRCF	<a href="#"><i>LL_I2C_ClearFlag_BERR</i></a>
	NACKCF	<a href="#"><i>LL_I2C_ClearFlag_NACK</i></a>

Register	Field	Function
ISR	OVRCF	<a href="#">LL_I2C_ClearFlag_OVR</a>
	PECCF	<a href="#">LL_I2C_ClearSMBusFlag_PECERR</a>
	STOPCF	<a href="#">LL_I2C_ClearFlag_STOP</a>
	TIMOUTCF	<a href="#">LL_I2C_ClearSMBusFlag_TIMEOUT</a>
ISR	ADD CODE	<a href="#">LL_I2C_GetAddressMatchCode</a>
	ADDR	<a href="#">LL_I2C_IsActiveFlag_ADDR</a>
	ALERT	<a href="#">LL_I2C_IsActiveSMBusFlag_ALERT</a>
	ARLO	<a href="#">LL_I2C_IsActiveFlag_ARLO</a>
	BERR	<a href="#">LL_I2C_IsActiveFlag_BERR</a>
	BUSY	<a href="#">LL_I2C_IsActiveFlag_BUSY</a>
	DIR	<a href="#">LL_I2C_GetTransferDirection</a>
	NACKF	<a href="#">LL_I2C_IsActiveFlag_NACK</a>
	OVR	<a href="#">LL_I2C_IsActiveFlag_OVR</a>
	PECERR	<a href="#">LL_I2C_IsActiveSMBusFlag_PECERR</a>
	RXNE	<a href="#">LL_I2C_IsActiveFlag_RXNE</a>
	STOPF	<a href="#">LL_I2C_IsActiveFlag_STOP</a>
	TC	<a href="#">LL_I2C_IsActiveFlag_TC</a>
	TCR	<a href="#">LL_I2C_IsActiveFlag_TCR</a>
	TIMEOUT	<a href="#">LL_I2C_IsActiveSMBusFlag_TIMEOUT</a>
	TXE	<a href="#">LL_I2C_ClearFlag_TXE</a>
		<a href="#">LL_I2C_IsActiveFlag_TXE</a>
	TXIS	<a href="#">LL_I2C_IsActiveFlag_TXIS</a>
OAR1	OA1	<a href="#">LL_I2C_SetOwnAddress1</a>
	OA1EN	<a href="#">LL_I2C_DisableOwnAddress1</a>
		<a href="#">LL_I2C_EnableOwnAddress1</a>
		<a href="#">LL_I2C_IsEnabledOwnAddress1</a>
	OA1MODE	<a href="#">LL_I2C_SetOwnAddress1</a>
OAR2	OA2	<a href="#">LL_I2C_SetOwnAddress2</a>
	OA2EN	<a href="#">LL_I2C_DisableOwnAddress2</a>
		<a href="#">LL_I2C_EnableOwnAddress2</a>
		<a href="#">LL_I2C_IsEnabledOwnAddress2</a>
	OA2MSK	<a href="#">LL_I2C_SetOwnAddress2</a>
PECR	PEC	<a href="#">LL_I2C_GetSMBusPEC</a>
RXDR	RXDATA	<a href="#">LL_I2C_DMA_GetRegAddr</a>
		<a href="#">LL_I2C_ReceiveData8</a>
TIMEOUTR	TEXTEN	<a href="#">LL_I2C_DisableSMBusTimeout</a>

Register	Field	Function
		<code>LL_I2C_EnableSMBusTimeout</code>
		<code>LL_I2C_IsEnabledSMBusTimeout</code>
	TIDLE	<code>LL_I2C_ConfigSMBusTimeout</code>
		<code>LL_I2C_GetSMBusTimeoutAMode</code>
	TIMEOUTA	<code>LL_I2C_SetSMBusTimeoutAMode</code>
		<code>LL_I2C_ConfigSMBusTimeout</code>
		<code>LL_I2C_GetSMBusTimeoutA</code>
		<code>LL_I2C_SetSMBusTimeoutA</code>
	TIMEOUTB	<code>LL_I2C_ConfigSMBusTimeout</code>
		<code>LL_I2C_GetSMBusTimeoutB</code>
		<code>LL_I2C_SetSMBusTimeoutB</code>
	TIMOUTEN	<code>LL_I2C_DisableSMBusTimeout</code>
		<code>LL_I2C_EnableSMBusTimeout</code>
		<code>LL_I2C_IsEnabledSMBusTimeout</code>
TIMINGR	PRESC	<code>LL_I2C_GetTimingPrescaler</code>
	SCLDEL	<code>LL_I2C_GetDataSetupTime</code>
	SCLH	<code>LL_I2C_GetClockHighPeriod</code>
	SCLL	<code>LL_I2C_GetClockLowPeriod</code>
	SDADEL	<code>LL_I2C_GetDataHoldTime</code>
	TIMINGR	<code>LL_I2C_SetTiming</code>
TXDR	TXDATA	<code>LL_I2C_DMA_GetRegAddr</code>
		<code>LL_I2C_TransmitData8</code>

## 78.12 I2S

Table 36: Correspondence between I2S registers and I2S low-layer driver functions

Register	Field	Function
CR2	ERRIE	<code>LL_I2S_DisableIT_ERR</code>
		<code>LL_I2S_EnableIT_ERR</code>
		<code>LL_I2S_IsEnabledIT_ERR</code>
	RXDMAEN	<code>LL_I2S_DisableDMAReq_RX</code>
		<code>LL_I2S_EnableDMAReq_RX</code>
		<code>LL_I2S_IsEnabledDMAReq_RX</code>
	RXNEIE	<code>LL_I2S_DisableIT_RXNE</code>
		<code>LL_I2S_EnableIT_RXNE</code>
		<code>LL_I2S_IsEnabledIT_RXNE</code>
	TXDMAEN	<code>LL_I2S_DisableDMAReq_TX</code>

Register	Field	Function
I2SCFGR	TXEIE	<a href="#">LL_I2S_EnableDMAReq_TX</a>
		<a href="#">LL_I2S_IsEnabledDMAReq_TX</a>
		<a href="#">LL_I2S_DisableIT_TXE</a>
		<a href="#">LL_I2S_EnableIT_TXE</a>
		<a href="#">LL_I2S_IsEnabledIT_TXE</a>
	DR	<a href="#">LL_I2S_ReceiveData16</a>
		<a href="#">LL_I2S_TransmitData16</a>
	ASTRTEN	<a href="#">LL_I2S_DisableAsyncStart</a>
		<a href="#">LL_I2S_EnableAsyncStart</a>
		<a href="#">LL_I2S_IsEnabledAsyncStart</a>
	CHLEN	<a href="#">LL_I2S_GetDataFormat</a>
		<a href="#">LL_I2S_SetDataFormat</a>
	CKPOL	<a href="#">LL_I2S_GetClockPolarity</a>
		<a href="#">LL_I2S_SetClockPolarity</a>
	DATLEN	<a href="#">LL_I2S_GetDataFormat</a>
		<a href="#">LL_I2S_SetDataFormat</a>
	I2SCFG	<a href="#">LL_I2S_GetTransferMode</a>
		<a href="#">LL_I2S_SetTransferMode</a>
	I2SE	<a href="#">LL_I2S_Disable</a>
		<a href="#">LL_I2S_Enable</a>
		<a href="#">LL_I2S_IsEnabled</a>
	I2SMOD	<a href="#">LL_I2S_Enable</a>
	I2SSTD	<a href="#">LL_I2S_GetStandard</a>
		<a href="#">LL_I2S_SetStandard</a>
	PCMSYNC	<a href="#">LL_I2S_GetStandard</a>
		<a href="#">LL_I2S_SetStandard</a>
I2SPR	I2SDIV	<a href="#">LL_I2S_GetPrescalerLinear</a>
		<a href="#">LL_I2S_SetPrescalerLinear</a>
	MCKOE	<a href="#">LL_I2S_DisableMasterClock</a>
		<a href="#">LL_I2S_EnableMasterClock</a>
		<a href="#">LL_I2S_IsEnabledMasterClock</a>
	ODD	<a href="#">LL_I2S_GetPrescalerParity</a>
		<a href="#">LL_I2S_SetPrescalerParity</a>
SR	BSY	<a href="#">LL_I2S_IsActiveFlag_BSY</a>
	CHSIDE	<a href="#">LL_I2S_IsActiveFlag_CHSIDE</a>
	FRE	<a href="#">LL_I2S_ClearFlag_FRE</a>

Register	Field	Function
	OVR	<a href="#">LL_I2S_IsActiveFlag_FRE</a>
		<a href="#">LL_I2S_ClearFlag_OVR</a>
		<a href="#">LL_I2S_IsActiveFlag_OVR</a>
	RXNE	<a href="#">LL_I2S_IsActiveFlag_RXNE</a>
	TXE	<a href="#">LL_I2S_IsActiveFlag_TXE</a>
	UDR	<a href="#">LL_I2S_ClearFlag_UDR</a>
		<a href="#">LL_I2S_IsActiveFlag_UDR</a>

## 78.13 IWDG

Table 37: Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
KR	KEY	<a href="#">LL_IWDG_DisableWriteAccess</a>
		<a href="#">LL_IWDG_Enable</a>
		<a href="#">LL_IWDG_EnableWriteAccess</a>
		<a href="#">LL_IWDG_ReloadCounter</a>
PR	PR	<a href="#">LL_IWDG_GetPrescaler</a>
		<a href="#">LL_IWDG_SetPrescaler</a>
RLR	RL	<a href="#">LL_IWDG_GetReloadCounter</a>
		<a href="#">LL_IWDG_SetReloadCounter</a>
SR	PVU	<a href="#">LL_IWDG_IsActiveFlag_PVU</a>
		<a href="#">LL_IWDG_IsReady</a>
	RVU	<a href="#">LL_IWDG_IsActiveFlag_RVU</a>
		<a href="#">LL_IWDG_IsReady</a>
	WVU	<a href="#">LL_IWDG_IsActiveFlag_WVU</a>
		<a href="#">LL_IWDG_IsReady</a>
WINR	WIN	<a href="#">LL_IWDG_GetWindow</a>
		<a href="#">LL_IWDG_SetWindow</a>

## 78.14 LPTIM

Table 38: Correspondence between LPTIM registers and LPTIM low-layer driver functions

Register	Field	Function
ARR	ARR	<a href="#">LL_LPTIM_GetAutoReload</a>
		<a href="#">LL_LPTIM_SetAutoReload</a>
CFGR	CKFLT	<a href="#">LL_LPTIM_ConfigClock</a>
		<a href="#">LL_LPTIM_GetClockFilter</a>

Register	Field	Function
	CKPOL	<a href="#"><i>LL_LPTIM_ConfigClock</i></a>
		<a href="#"><i>LL_LPTIM_GetClockPolarity</i></a>
		<a href="#"><i>LL_LPTIM_GetEncoderMode</i></a>
		<a href="#"><i>LL_LPTIM_SetEncoderMode</i></a>
	CKSEL	<a href="#"><i>LL_LPTIM_GetClockSource</i></a>
		<a href="#"><i>LL_LPTIM_SetClockSource</i></a>
	COUNTMODE	<a href="#"><i>LL_LPTIM_GetCounterMode</i></a>
		<a href="#"><i>LL_LPTIM_SetCounterMode</i></a>
	ENC	<a href="#"><i>LL_LPTIM_DisableEncoderMode</i></a>
		<a href="#"><i>LL_LPTIM_EnableEncoderMode</i></a>
		<a href="#"><i>LL_LPTIM_IsEnabledEncoderMode</i></a>
	PRELOAD	<a href="#"><i>LL_LPTIM_GetUpdateMode</i></a>
		<a href="#"><i>LL_LPTIM_SetUpdateMode</i></a>
	PRESC	<a href="#"><i>LL_LPTIM_GetPrescaler</i></a>
		<a href="#"><i>LL_LPTIM_SetPrescaler</i></a>
	TIMOUT	<a href="#"><i>LL_LPTIM_DisableTimeout</i></a>
		<a href="#"><i>LL_LPTIM_EnableTimeout</i></a>
		<a href="#"><i>LL_LPTIM_IsEnabledTimeout</i></a>
	TRGFLT	<a href="#"><i>LL_LPTIM_ConfigTrigger</i></a>
		<a href="#"><i>LL_LPTIM_GetTriggerFilter</i></a>
	TRIGEN	<a href="#"><i>LL_LPTIM_ConfigTrigger</i></a>
		<a href="#"><i>LL_LPTIM_GetTriggerPolarity</i></a>
		<a href="#"><i>LL_LPTIM_TrigSw</i></a>
	TRIGSEL	<a href="#"><i>LL_LPTIM_ConfigTrigger</i></a>
		<a href="#"><i>LL_LPTIM_GetTriggerSource</i></a>
	WAVE	<a href="#"><i>LL_LPTIM_ConfigOutput</i></a>
		<a href="#"><i>LL_LPTIM_GetWaveform</i></a>
		<a href="#"><i>LL_LPTIM_SetWaveform</i></a>
	WAVPOL	<a href="#"><i>LL_LPTIM_ConfigOutput</i></a>
		<a href="#"><i>LL_LPTIM_GetPolarity</i></a>
		<a href="#"><i>LL_LPTIM_SetPolarity</i></a>
CMP	CMP	<a href="#"><i>LL_LPTIM_GetCompare</i></a>
		<a href="#"><i>LL_LPTIM_SetCompare</i></a>
CNT	CNT	<a href="#"><i>LL_LPTIM_GetCounter</i></a>
CR	CNTSTRT	<a href="#"><i>LL_LPTIM_StartCounter</i></a>
	ENABLE	<a href="#"><i>LL_LPTIM_Disable</i></a>

Register	Field	Function
		<i>LL_LPTIM_Enable</i>
		<i>LL_LPTIM_IsEnabled</i>
	SNGSTRT	<i>LL_LPTIM_StartCounter</i>
ICR	ARRMCF	<i>LL_LPTIM_ClearFLAG_ARRM</i>
	ARROKCF	<i>LL_LPTIM_ClearFlag_ARROK</i>
	CMPMCF	<i>LL_LPTIM_ClearFLAG_CMPM</i>
	CMPOKCF	<i>LL_LPTIM_ClearFlag_CMPOK</i>
	DOWNCF	<i>LL_LPTIM_ClearFlag_DOWN</i>
	EXTTRIGCF	<i>LL_LPTIM_ClearFlag_EXTTRIG</i>
	UPCF	<i>LL_LPTIM_ClearFlag_UP</i>
IER	ARRMIE	<i>LL_LPTIM_DisableIT_ARRM</i>
		<i>LL_LPTIM_EnableIT_ARRM</i>
		<i>LL_LPTIM_IsEnabledIT_ARRM</i>
	ARROKIE	<i>LL_LPTIM_DisableIT_ARROK</i>
		<i>LL_LPTIM_EnableIT_ARROK</i>
		<i>LL_LPTIM_IsEnabledIT_ARROK</i>
	CMPMIE	<i>LL_LPTIM_DisableIT_CMPM</i>
		<i>LL_LPTIM_EnableIT_CMPM</i>
		<i>LL_LPTIM_IsEnabledIT_CMPM</i>
	CMPOKIE	<i>LL_LPTIM_DisableIT_CMPOK</i>
		<i>LL_LPTIM_EnableIT_CMPOK</i>
		<i>LL_LPTIM_IsEnabledIT_CMPOK</i>
	DOWNIE	<i>LL_LPTIM_DisableIT_DOWN</i>
		<i>LL_LPTIM_EnableIT_DOWN</i>
		<i>LL_LPTIM_IsEnabledIT_DOWN</i>
	EXTTRIGIE	<i>LL_LPTIM_DisableIT_EXTTRIG</i>
		<i>LL_LPTIM_EnableIT_EXTTRIG</i>
		<i>LL_LPTIM_IsEnabledIT_EXTTRIG</i>
	UPIE	<i>LL_LPTIM_DisableIT_UP</i>
		<i>LL_LPTIM_EnableIT_UP</i>
		<i>LL_LPTIM_IsEnabledIT_UP</i>
ISR	ARRM	<i>LL_LPTIM_IsActiveFlag_ARRM</i>
	ARROK	<i>LL_LPTIM_IsActiveFlag_ARROK</i>
	CMPM	<i>LL_LPTIM_IsActiveFlag_CMPM</i>
	CMPOK	<i>LL_LPTIM_IsActiveFlag_CMPOK</i>
	DOWN	<i>LL_LPTIM_IsActiveFlag_DOWN</i>

Register	Field	Function
	EXTTRIG	<i>LL_LPTIM_IsActiveFlag_EXTTRIG</i>
	UP	<i>LL_LPTIM_IsActiveFlag_UP</i>

## 78.15 LPUART

Table 39: Correspondence between LPUART registers and LPUART low-layer driver functions

Register	Field	Function
BRR	BRR	<i>LL_LPUART_GetBaudRate</i>
		<i>LL_LPUART_SetBaudRate</i>
	CMIE	<i>LL_LPUART_DisableIT_CM</i>
		<i>LL_LPUART_EnableIT_CM</i>
		<i>LL_LPUART_IsEnabledIT_CM</i>
	DEAT	<i>LL_LPUART_GetDEAssertionTime</i>
		<i>LL_LPUART_SetDEAssertionTime</i>
	DEDT	<i>LL_LPUART_GetDEDeassertionTime</i>
		<i>LL_LPUART_SetDEDeassertionTime</i>
	IDLEIE	<i>LL_LPUART_DisableIT_IDLE</i>
		<i>LL_LPUART_EnableIT_IDLE</i>
		<i>LL_LPUART_IsEnabledIT_IDLE</i>
CR1	M	<i>LL_LPUART_ConfigCharacter</i>
		<i>LL_LPUART_GetDataWidth</i>
		<i>LL_LPUART_SetDataWidth</i>
	MME	<i>LL_LPUART_DisableMuteMode</i>
		<i>LL_LPUART_EnableMuteMode</i>
		<i>LL_LPUART_IsEnabledMuteMode</i>
	PCE	<i>LL_LPUART_ConfigCharacter</i>
		<i>LL_LPUART_GetParity</i>
		<i>LL_LPUART_SetParity</i>
	PEIE	<i>LL_LPUART_DisableIT_PE</i>
		<i>LL_LPUART_EnableIT_PE</i>
		<i>LL_LPUART_IsEnabledIT_PE</i>
RE	PS	<i>LL_LPUART_ConfigCharacter</i>
		<i>LL_LPUART_GetParity</i>
		<i>LL_LPUART_SetParity</i>
	RE	<i>LL_LPUART_DisableDirectionRx</i>
		<i>LL_LPUART_EnableDirectionRx</i>
		<i>LL_LPUART_GetTransferDirection</i>

Register	Field	Function
	RXNEIE	<i>LL_LPUART_SetTransferDirection</i>
		<i>LL_LPUART_DisableIT_RXNE</i>
		<i>LL_LPUART_EnableIT_RXNE</i>
		<i>LL_LPUART_IsEnabledIT_RXNE</i>
	TCIE	<i>LL_LPUART_DisableIT_TC</i>
		<i>LL_LPUART_EnableIT_TC</i>
		<i>LL_LPUART_IsEnabledIT_TC</i>
	TE	<i>LL_LPUART_DisableDirectionTx</i>
		<i>LL_LPUART_EnableDirectionTx</i>
		<i>LL_LPUART_GetTransferDirection</i>
		<i>LL_LPUART_SetTransferDirection</i>
	TXEIE	<i>LL_LPUART_DisableIT_TXE</i>
		<i>LL_LPUART_EnableIT_TXE</i>
		<i>LL_LPUART_IsEnabledIT_TXE</i>
	UE	<i>LL_LPUART_Disable</i>
		<i>LL_LPUART_Enable</i>
		<i>LL_LPUART_IsEnabled</i>
	UESM	<i>LL_LPUART_DisableInStopMode</i>
		<i>LL_LPUART_EnableInStopMode</i>
		<i>LL_LPUART_IsEnabledInStopMode</i>
	WAKE	<i>LL_LPUART_GetWakeUpMethod</i>
		<i>LL_LPUART_SetWakeUpMethod</i>
CR2	ADD	<i>LL_LPUART_ConfigNodeAddress</i>
		<i>LL_LPUART_GetNodeAddress</i>
	ADDM7	<i>LL_LPUART_ConfigNodeAddress</i>
		<i>LL_LPUART_GetNodeAddressLen</i>
	DATAINV	<i>LL_LPUART_GetBinaryDataLogic</i>
		<i>LL_LPUART_SetBinaryDataLogic</i>
	MSBFIRST	<i>LL_LPUART_GetTransferBitOrder</i>
		<i>LL_LPUART_SetTransferBitOrder</i>
	RXINV	<i>LL_LPUART_GetRXPinLevel</i>
		<i>LL_LPUART_SetRXPinLevel</i>
	STOP	<i>LL_LPUART_ConfigCharacter</i>
		<i>LL_LPUART_GetStopBitsLength</i>
		<i>LL_LPUART_SetStopBitsLength</i>
	SWAP	<i>LL_LPUART_GetTXRXSwap</i>

Register	Field	Function
CR3	TXINV	<i>LL_LPUART_SetTXRXSwap</i>
		<i>LL_LPUART_GetTXPinLevel</i>
		<i>LL_LPUART_SetTXPinLevel</i>
	CTSE	<i>LL_LPUART_DisableCTSHWFlowCtrl</i>
		<i>LL_LPUART_EnableCTSHWFlowCtrl</i>
		<i>LL_LPUART_GetHWFlowCtrl</i>
		<i>LL_LPUART_SetHWFlowCtrl</i>
	CTSIE	<i>LL_LPUART_DisableIT_CTS</i>
		<i>LL_LPUART_EnableIT_CTS</i>
		<i>LL_LPUART_IsEnabledIT_CTS</i>
	DDRE	<i>LL_LPUART_DisableDMAactOnRxErr</i>
		<i>LL_LPUART_EnableDMAactOnRxErr</i>
		<i>LL_LPUART_IsEnabledDMAactOnRxErr</i>
	DEM	<i>LL_LPUART_DisableDEMode</i>
		<i>LL_LPUART_EnableDEMode</i>
		<i>LL_LPUART_IsEnabledDEMode</i>
	DEP	<i>LL_LPUART_GetDESignalPolarity</i>
		<i>LL_LPUART_SetDESignalPolarity</i>
	DMAR	<i>LL_LPUART_DisableDMAReq_RX</i>
		<i>LL_LPUART_EnableDMAReq_RX</i>
		<i>LL_LPUART_IsEnabledDMAReq_RX</i>
	DMAT	<i>LL_LPUART_DisableDMAReq_TX</i>
		<i>LL_LPUART_EnableDMAReq_TX</i>
		<i>LL_LPUART_IsEnabledDMAReq_TX</i>
	EIE	<i>LL_LPUART_DisableIT_ERROR</i>
		<i>LL_LPUART_EnableIT_ERROR</i>
		<i>LL_LPUART_IsEnabledIT_ERROR</i>
	HDSEL	<i>LL_LPUART_DisableHalfDuplex</i>
		<i>LL_LPUART_EnableHalfDuplex</i>
		<i>LL_LPUART_IsEnabledHalfDuplex</i>
	OVRDIS	<i>LL_LPUART_DisableOverrunDetect</i>
		<i>LL_LPUART_EnableOverrunDetect</i>
		<i>LL_LPUART_IsEnabledOverrunDetect</i>
	RTSE	<i>LL_LPUART_DisableRTSHWFlowCtrl</i>
		<i>LL_LPUART_EnableRTSHWFlowCtrl</i>
		<i>LL_LPUART_GetHWFlowCtrl</i>

Register	Field	Function
WUFIE		<i>LL_LPUART_SetHWFlowCtrl</i>
		<i>LL_LPUART_DisableIT_WKUP</i>
		<i>LL_LPUART_EnableIT_WKUP</i>
		<i>LL_LPUART_IsEnabledIT_WKUP</i>
	WUS	<i>LL_LPUART_GetWKUPType</i>
		<i>LL_LPUART_SetWKUPType</i>
ICR	CMCF	<i>LL_LPUART_ClearFlag_CM</i>
	CTSCF	<i>LL_LPUART_ClearFlag_nCTS</i>
	FECF	<i>LL_LPUART_ClearFlag_FE</i>
	IDLECF	<i>LL_LPUART_ClearFlag_IDLE</i>
	NCF	<i>LL_LPUART_ClearFlag_NE</i>
	ORECF	<i>LL_LPUART_ClearFlag_ORE</i>
	PECF	<i>LL_LPUART_ClearFlag_PE</i>
	TCCF	<i>LL_LPUART_ClearFlag_TC</i>
	WUCF	<i>LL_LPUART_ClearFlag_WKUP</i>
ISR	BUSY	<i>LL_LPUART_IsActiveFlag_BUSY</i>
	CMF	<i>LL_LPUART_IsActiveFlag_CM</i>
	CTS	<i>LL_LPUART_IsActiveFlag_CTS</i>
	CTSIF	<i>LL_LPUART_IsActiveFlag_nCTS</i>
	FE	<i>LL_LPUART_IsActiveFlag_FE</i>
	IDLE	<i>LL_LPUART_IsActiveFlag_IDLE</i>
	NE	<i>LL_LPUART_IsActiveFlag_NE</i>
	ORE	<i>LL_LPUART_IsActiveFlag_ORE</i>
	PE	<i>LL_LPUART_IsActiveFlag_PE</i>
	REACK	<i>LL_LPUART_IsActiveFlag_REACK</i>
	RWU	<i>LL_LPUART_IsActiveFlag_RWU</i>
	RXNE	<i>LL_LPUART_IsActiveFlag_RXNE</i>
	SBKF	<i>LL_LPUART_IsActiveFlag_SBK</i>
	TC	<i>LL_LPUART_IsActiveFlag_TC</i>
	TEACK	<i>LL_LPUART_IsActiveFlag_TEACK</i>
	TXE	<i>LL_LPUART_IsActiveFlag_TXE</i>
	WUF	<i>LL_LPUART_IsActiveFlag_WKUP</i>
RDR	RDR	<i>LL_LPUART_DMA_GetRegAddr</i>
		<i>LL_LPUART_ReceiveData8</i>
		<i>LL_LPUART_ReceiveData9</i>
RQR	MMRQ	<i>LL_LPUART_RequestEnterMuteMode</i>

Register	Field	Function
	RXFRQ	<i>LL_LPUART_RequestRxDataFlush</i>
	SBKRQ	<i>LL_LPUART_RequestBreakSending</i>
TDR	TDR	<i>LL_LPUART_DMA_GetRegAddr</i>
		<i>LL_LPUART_TransmitData8</i>
		<i>LL_LPUART_TransmitData9</i>

## 78.16 PWR

Table 40: Correspondence between PWR registers and PWR low-layer driver functions

Register	Field	Function
CR	CSBF	<i>LL_PWR_ClearFlag_SB</i>
	CWUF	<i>LL_PWR_ClearFlag_WU</i>
	DBP	<i>LL_PWR_DisableBkUpAccess</i>
		<i>LL_PWR_EnableBkUpAccess</i>
		<i>LL_PWR_IsEnabledBkUpAccess</i>
	DS_EE_KOFF	<i>LL_PWR_DisableNVMKeptOff</i>
		<i>LL_PWR_EnableNVMKeptOff</i>
		<i>LL_PWR_IsEnabledNVMKeptOff</i>
	FWU	<i>LL_PWR_DisableFastWakeUp</i>
		<i>LL_PWR_EnableFastWakeUp</i>
		<i>LL_PWR_IsEnabledFastWakeUp</i>
	LPRUN	<i>LL_PWR_DisableLowPowerRunMode</i>
		<i>LL_PWR_EnableLowPowerRunMode</i>
		<i>LL_PWR_EnterLowPowerRunMode</i>
		<i>LL_PWR_ExitLowPowerRunMode</i>
		<i>LL_PWR_IsEnabledLowPowerRunMode</i>
	LPDDR	<i>LL_PWR_EnterLowPowerRunMode</i>
		<i>LL_PWR_ExitLowPowerRunMode</i>
		<i>LL_PWR_GetRegulModeLP</i>
		<i>LL_PWR_SetRegulModeLP</i>
	PDDS	<i>LL_PWR_GetPowerMode</i>
		<i>LL_PWR_SetPowerMode</i>
	PLS	<i>LL_PWR_GetPVDLevel</i>
		<i>LL_PWR_SetPVDLevel</i>
	PVDE	<i>LL_PWR_DisablePVD</i>
		<i>LL_PWR_EnablePVD</i>
		<i>LL_PWR_IsEnabledPVD</i>

Register	Field	Function
CSR	ULP	<i>LL_PWR_DisableUltraLowPower</i>
		<i>LL_PWR_EnableUltraLowPower</i>
		<i>LL_PWR_IsEnabledUltraLowPower</i>
	VOS	<i>LL_PWR_GetRegulVoltageScaling</i>
		<i>LL_PWR_SetRegulVoltageScaling</i>
	EWUP1	<i>LL_PWR_DisableWakeUpPin</i>
		<i>LL_PWR_EnableWakeUpPin</i>
		<i>LL_PWR_IsEnabledWakeUpPin</i>
	EWUP2	<i>LL_PWR_DisableWakeUpPin</i>
		<i>LL_PWR_EnableWakeUpPin</i>
		<i>LL_PWR_IsEnabledWakeUpPin</i>
	EWUP3	<i>LL_PWR_DisableWakeUpPin</i>
		<i>LL_PWR_EnableWakeUpPin</i>
		<i>LL_PWR_IsEnabledWakeUpPin</i>
	PVDO	<i>LL_PWR_IsActiveFlag_PVDO</i>
	REGLPF	<i>LL_PWR_IsActiveFlag_REGLPF</i>
	SBF	<i>LL_PWR_IsActiveFlag_SB</i>
	VOSF	<i>LL_PWR_IsActiveFlag_VOSF</i>
	VREFINTRDYF	<i>LL_PWR_IsActiveFlag_VREFINTRDY</i>
	WUF	<i>LL_PWR_IsActiveFlag_WU</i>

## 78.17 RCC

Table 41: Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
CCIPR	CLK48SEL	<i>LL_RCC_GetRNGClockSource</i>
		<i>LL_RCC_SetRNGClockSource</i>
	HSI48SEL	<i>LL_RCC_SetRNGClockSource</i>
		<i>LL_RCC_SetUSBClockSource</i>
	I2CxSEL	<i>LL_RCC_GetI2CClockSource</i>
		<i>LL_RCC_SetI2CClockSource</i>
	LPTIMxSEL	<i>LL_RCC_GetLPTIMClockSource</i>
		<i>LL_RCC_SetLPTIMClockSource</i>
	LPUART1SEL	<i>LL_RCC_GetLPUARTClockSource</i>
		<i>LL_RCC_SetLPUARTClockSource</i>
	USARTxSEL	<i>LL_RCC_GetUSARTClockSource</i>
		<i>LL_RCC_SetUSARTClockSource</i>

Register	Field	Function
CFGR	HPRE	<code>LL_RCC_GetAHBPrescaler</code>
		<code>LL_RCC_SetAHBPrescaler</code>
	MCOPRE	<code>LL_RCC_ConfigMCO</code>
	MCOSEL	<code>LL_RCC_ConfigMCO</code>
	PLLDIV	<code>LL_RCC_PLL_ConfigDomain_SYS</code>
		<code>LL_RCC_PLL_GetDivider</code>
	PLLMUL	<code>LL_RCC_PLL_ConfigDomain_SYS</code>
		<code>LL_RCC_PLL_GetMultiplicator</code>
	PLLCSR	<code>LL_RCC_PLL_ConfigDomain_SYS</code>
		<code>LL_RCC_PLL_GetMainSource</code>
	PPRE1	<code>LL_RCC_GetAPB1Prescaler</code>
		<code>LL_RCC_SetAPB1Prescaler</code>
	PPRE2	<code>LL_RCC_GetAPB2Prescaler</code>
		<code>LL_RCC_SetAPB2Prescaler</code>
	STOPWUCK	<code>LL_RCC_GetClkAfterWakeFromStop</code>
		<code>LL_RCC_SetClkAfterWakeFromStop</code>
	SW	<code>LL_RCC_SetSysClkSource</code>
	SWS	<code>LL_RCC_GetSysClkSource</code>
CICR	CSSC	<code>LL_RCC_ClearFlag_HSECSS</code>
	HSERDYC	<code>LL_RCC_ClearFlag_HSERDY</code>
	HSI48RDYC	<code>LL_RCC_ClearFlag_HSI48RDY</code>
	HSIRDYC	<code>LL_RCC_ClearFlag_HSIRDY</code>
	LSECSSC	<code>LL_RCC_ClearFlag_LSECSS</code>
	LSERDYC	<code>LL_RCC_ClearFlag_LSERDY</code>
	LSIRDYC	<code>LL_RCC_ClearFlag_LSIRDY</code>
	MSIRDYC	<code>LL_RCC_ClearFlag_MSIRDY</code>
	PLLRDYC	<code>LL_RCC_ClearFlag_PLLRDY</code>
CIER	HSERDYIE	<code>LL_RCC_DisableIT_HSERDY</code>
		<code>LL_RCC_EnableIT_HSERDY</code>
		<code>LL_RCC_IsEnabledIT_HSERDY</code>
	HSI48RDYIE	<code>LL_RCC_DisableIT_HSI48RDY</code>
		<code>LL_RCC_EnableIT_HSI48RDY</code>
		<code>LL_RCC_IsEnabledIT_HSI48RDY</code>
	HSIRDYIE	<code>LL_RCC_DisableIT_HSIRDY</code>
		<code>LL_RCC_EnableIT_HSIRDY</code>
		<code>LL_RCC_IsEnabledIT_HSIRDY</code>

Register	Field	Function
	LSECSSIE	<i>LL_RCC_DisableIT_LSECSS</i>
		<i>LL_RCC_EnableIT_LSECSS</i>
		<i>LL_RCC_IsEnabledIT_LSECSS</i>
	LSERDYIE	<i>LL_RCC_DisableIT_LSERDY</i>
		<i>LL_RCC_EnableIT_LSERDY</i>
		<i>LL_RCC_IsEnabledIT_LSERDY</i>
	LSIRDYIE	<i>LL_RCC_DisableIT_LSIRDY</i>
		<i>LL_RCC_EnableIT_LSIRDY</i>
		<i>LL_RCC_IsEnabledIT_LSIRDY</i>
	MSIRDYIE	<i>LL_RCC_DisableIT_MSIRDY</i>
		<i>LL_RCC_EnableIT_MSIRDY</i>
		<i>LL_RCC_IsEnabledIT_MSIRDY</i>
	PLLRDYIE	<i>LL_RCC_DisableIT_PLLRDY</i>
		<i>LL_RCC_EnableIT_PLLRDY</i>
		<i>LL_RCC_IsEnabledIT_PLLRDY</i>
CIFR	CSSF	<i>LL_RCC_IsActiveFlag_HSECSS</i>
	HSDRDYF	<i>LL_RCC_IsActiveFlag_HSERDY</i>
	HSI48RDYF	<i>LL_RCC_IsActiveFlag_HSI48RDY</i>
	HSIRDYF	<i>LL_RCC_IsActiveFlag_HSIRDY</i>
	LSECSSF	<i>LL_RCC_IsActiveFlag_LSECSS</i>
	LSERDYF	<i>LL_RCC_IsActiveFlag_LSERDY</i>
	LSIRDYF	<i>LL_RCC_IsActiveFlag_LSIRDY</i>
	MSIRDYF	<i>LL_RCC_IsActiveFlag_MSIRDY</i>
	PLLRDYF	<i>LL_RCC_IsActiveFlag_PLLRDY</i>
CR	CSSHSEON	<i>LL_RCC_HSE_EnableCSS</i>
	HSEBYP	<i>LL_RCC_HSE_DisableBypass</i>
		<i>LL_RCC_HSE_EnableBypass</i>
	HSEON	<i>LL_RCC_HSE_Disable</i>
		<i>LL_RCC_HSE_Enable</i>
	HSDRDY	<i>LL_RCC_HSE_IsReady</i>
	HSIDIVEN	<i>LL_RCC_HSI_DisableDivider</i>
		<i>LL_RCC_HSI_EnableDivider</i>
	HSIDIVF	<i>LL_RCC_IsActiveFlag_HSIDIV</i>
	HSIKERON	<i>LL_RCC_HSI_DisableInStopMode</i>
		<i>LL_RCC_HSI_EnableInStopMode</i>
	HSION	<i>LL_RCC_HSI_Disable</i>

Register	Field	Function
CRCCR	HSIOUTEN	<i>LL_RCC_HSI_Enable</i>
		<i>LL_RCC_HSI_DisableOutput</i>
		<i>LL_RCC_HSI_EnableOutput</i>
	HSIRDY	<i>LL_RCC_HSI_IsReady</i>
	MSION	<i>LL_RCC_MSI_Disable</i>
		<i>LL_RCC_MSI_Enable</i>
	MSIRDY	<i>LL_RCC_MSI_IsReady</i>
	PLLON	<i>LL_RCC_PLL_Disable</i>
		<i>LL_RCC_PLL_Enable</i>
	PLLRDY	<i>LL_RCC_PLL_IsReady</i>
CSR	RTCPRE	<i>LL_RCC_GetRTC_HSEPrescaler</i>
		<i>LL_RCC_SetRTC_HSEPrescaler</i>
	HSI48CAL	<i>LL_RCC_HSI48_GetCalibration</i>
	HSI48DIV6OUTEN	<i>LL_RCC_HSI48_DisableDivider</i>
		<i>LL_RCC_HSI48_EnableDivider</i>
		<i>LL_RCC_HSI48_IsDivided</i>
	HSI48ON	<i>LL_RCC_HSI48_Disable</i>
		<i>LL_RCC_HSI48_Enable</i>
	HSI48RDY	<i>LL_RCC_HSI48_IsReady</i>
	FWRSTF	<i>LL_RCC_IsActiveFlag_FWRST</i>
	IWDGRSTF	<i>LL_RCC_IsActiveFlag_IWDGRST</i>
	LPWRRSTF	<i>LL_RCC_IsActiveFlag_LPWRRST</i>
	LSEBYP	<i>LL_RCC_LSE_DisableBypass</i>
		<i>LL_RCC_LSE_EnableBypass</i>
	LSECSSD	<i>LL_RCC_LSE_IsCSSDetected</i>
	LSECSSON	<i>LL_RCC_LSE_DisableCSS</i>
		<i>LL_RCC_LSE_EnableCSS</i>
	LSEDRV	<i>LL_RCC_LSE_GetDriveCapability</i>
		<i>LL_RCC_LSE_SetDriveCapability</i>
	LSEON	<i>LL_RCC_LSE_Disable</i>
		<i>LL_RCC_LSE_Enable</i>
	LSERDY	<i>LL_RCC_LSE_IsReady</i>
	LSION	<i>LL_RCC_LSI_Disable</i>
		<i>LL_RCC_LSI_Enable</i>
	LSIRDY	<i>LL_RCC_LSI_IsReady</i>
	OBLRSTF	<i>LL_RCC_IsActiveFlag_OBLRST</i>

Register	Field	Function
ICSCR	PINRSTF	<i>LL_RCC_IsActiveFlag_PINRST</i>
	PORRSTF	<i>LL_RCC_IsActiveFlag_PORRST</i>
	RMVF	<i>LL_RCC_ClearResetFlags</i>
	RTCEN	<i>LL_RCC_DisableRTC</i>
		<i>LL_RCC_EnableRTC</i>
		<i>LL_RCC_IsEnabledRTC</i>
	RTCRST	<i>LL_RCC_ForceBackupDomainReset</i>
		<i>LL_RCC_ReleaseBackupDomainReset</i>
	RTCSEL	<i>LL_RCC_GetRTCClockSource</i>
		<i>LL_RCC_SetRTCClockSource</i>
	SFTRSTF	<i>LL_RCC_IsActiveFlag_SFTRST</i>
	WWDGRSTF	<i>LL_RCC_IsActiveFlag_WWDGRST</i>
ICSCR	HSICAL	<i>LL_RCC_HSI_GetCalibration</i>
	HSITRIM	<i>LL_RCC_HSI_GetCalibTrimming</i>
		<i>LL_RCC_HSI_SetCalibTrimming</i>
	MSICAL	<i>LL_RCC_MSI_GetCalibration</i>
	MSIRANGE	<i>LL_RCC_MSI_GetRange</i>
		<i>LL_RCC_MSI_SetRange</i>
	MSITRIM	<i>LL_RCC_MSI_GetCalibTrimming</i>
		<i>LL_RCC_MSI_SetCalibTrimming</i>

## 78.18 RNG

Table 42: Correspondence between RNG registers and RNG low-layer driver functions

Register	Field	Function
CR	IE	<i>LL_RNG_DisableIT</i>
		<i>LL_RNG_EnableIT</i>
		<i>LL_RNG_IsEnabledIT</i>
	RNGEN	<i>LL_RNG_Disable</i>
		<i>LL_RNG_Enable</i>
		<i>LL_RNG_IsEnabled</i>
DR	RNDATA	<i>LL_RNG_ReadRandData32</i>
SR	CECS	<i>LL_RNG_IsActiveFlag_CECS</i>
	CEIS	<i>LL_RNG_ClearFlag_CEIS</i>
		<i>LL_RNG_IsActiveFlag_CEIS</i>
	DRDY	<i>LL_RNG_IsActiveFlag_DRDY</i>
	SECS	<i>LL_RNG_IsActiveFlag_SECS</i>

Register	Field	Function
	SEIS	<i>LL_RNG_ClearFlag_SEIS</i>
		<i>LL_RNG_IsActiveFlag_SEIS</i>

## 78.19 RTC

Table 43: Correspondence between RTC registers and RTC low-layer driver functions

Register	Field	Function
ALRMAR	DT	<i>LL_RTC_ALMA_GetDay</i>
		<i>LL_RTC_ALMA_SetDay</i>
	DU	<i>LL_RTC_ALMA_GetDay</i>
		<i>LL_RTC_ALMA_GetWeekDay</i>
		<i>LL_RTC_ALMA_SetDay</i>
		<i>LL_RTC_ALMA_SetWeekDay</i>
	HT	<i>LL_RTC_ALMA_ConfigTime</i>
		<i>LL_RTC_ALMA_GetHour</i>
		<i>LL_RTC_ALMA_GetTime</i>
		<i>LL_RTC_ALMA_SetHour</i>
	HU	<i>LL_RTC_ALMA_ConfigTime</i>
		<i>LL_RTC_ALMA_GetHour</i>
		<i>LL_RTC_ALMA_GetTime</i>
		<i>LL_RTC_ALMA_SetHour</i>
	MNT	<i>LL_RTC_ALMA_ConfigTime</i>
		<i>LL_RTC_ALMA_GetMinute</i>
		<i>LL_RTC_ALMA_GetTime</i>
		<i>LL_RTC_ALMA_SetMinute</i>
	MNU	<i>LL_RTC_ALMA_ConfigTime</i>
		<i>LL_RTC_ALMA_GetMinute</i>
		<i>LL_RTC_ALMA_GetTime</i>
		<i>LL_RTC_ALMA_SetMinute</i>
	MSK1	<i>LL_RTC_ALMA_GetMask</i>
		<i>LL_RTC_ALMA_SetMask</i>
	MSK2	<i>LL_RTC_ALMA_GetMask</i>
		<i>LL_RTC_ALMA_SetMask</i>
	MSK3	<i>LL_RTC_ALMA_GetMask</i>
		<i>LL_RTC_ALMA_SetMask</i>
	MSK4	<i>LL_RTC_ALMA_GetMask</i>
		<i>LL_RTC_ALMA_SetMask</i>

Register	Field	Function
ALRMASSR	PM	<a href="#"><i>LL_RTC_ALMA_ConfigTime</i></a>
		<a href="#"><i>LL_RTC_ALMA_GetTimeFormat</i></a>
		<a href="#"><i>LL_RTC_ALMA_SetTimeFormat</i></a>
	ST	<a href="#"><i>LL_RTC_ALMA_ConfigTime</i></a>
		<a href="#"><i>LL_RTC_ALMA_GetSecond</i></a>
		<a href="#"><i>LL_RTC_ALMA_GetTime</i></a>
		<a href="#"><i>LL_RTC_ALMA_SetSecond</i></a>
	SU	<a href="#"><i>LL_RTC_ALMA_ConfigTime</i></a>
		<a href="#"><i>LL_RTC_ALMA_GetSecond</i></a>
		<a href="#"><i>LL_RTC_ALMA_GetTime</i></a>
		<a href="#"><i>LL_RTC_ALMA_SetSecond</i></a>
	WDSEL	<a href="#"><i>LL_RTC_ALMA_DisableWeekday</i></a>
		<a href="#"><i>LL_RTC_ALMA_EnableWeekday</i></a>
	MASKSS	<a href="#"><i>LL_RTC_ALMA_GetSubSecondMask</i></a>
		<a href="#"><i>LL_RTC_ALMA_SetSubSecondMask</i></a>
	SS	<a href="#"><i>LL_RTC_ALMA_GetSubSecond</i></a>
		<a href="#"><i>LL_RTC_ALMA_SetSubSecond</i></a>
ALRMBR	DT	<a href="#"><i>LL_RTC_ALMB_GetDay</i></a>
		<a href="#"><i>LL_RTC_ALMB_SetDay</i></a>
	DU	<a href="#"><i>LL_RTC_ALMB_GetDay</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetWeekDay</i></a>
		<a href="#"><i>LL_RTC_ALMB_SetDay</i></a>
		<a href="#"><i>LL_RTC_ALMB_SetWeekDay</i></a>
	HT	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetHour</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetTime</i></a>
		<a href="#"><i>LL_RTC_ALMB_SetHour</i></a>
	HU	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetHour</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetTime</i></a>
		<a href="#"><i>LL_RTC_ALMB_SetHour</i></a>
	MNT	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetMinute</i></a>
		<a href="#"><i>LL_RTC_ALMB_GetTime</i></a>
		<a href="#"><i>LL_RTC_ALMB_SetMinute</i></a>
	MNU	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a>

Register	Field	Function
		<a href="#"><i>LL_RTC_ALMB_GetMinute</i></a> <a href="#"><i>LL_RTC_ALMB_SetMinute</i></a>
	MSK1	<a href="#"><i>LL_RTC_ALMB_GetMask</i></a> <a href="#"><i>LL_RTC_ALMB_SetMask</i></a>
	MSK2	<a href="#"><i>LL_RTC_ALMB_GetMask</i></a> <a href="#"><i>LL_RTC_ALMB_SetMask</i></a>
	MSK3	<a href="#"><i>LL_RTC_ALMB_GetMask</i></a> <a href="#"><i>LL_RTC_ALMB_SetMask</i></a>
	MSK4	<a href="#"><i>LL_RTC_ALMB_GetMask</i></a> <a href="#"><i>LL_RTC_ALMB_SetMask</i></a>
	PM	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a> <a href="#"><i>LL_RTC_ALMB_GetTimeFormat</i></a> <a href="#"><i>LL_RTC_ALMB_SetTimeFormat</i></a>
	ST	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a> <a href="#"><i>LL_RTC_ALMB_GetSecond</i></a> <a href="#"><i>LL_RTC_ALMB_SetSecond</i></a>
	SU	<a href="#"><i>LL_RTC_ALMB_ConfigTime</i></a> <a href="#"><i>LL_RTC_ALMB_SetSecond</i></a>
	WDSEL	<a href="#"><i>LL_RTC_ALMB_DisableWeekday</i></a> <a href="#"><i>LL_RTC_ALMB_EnableWeekday</i></a>
ALRMBSSR	MASKSS	<a href="#"><i>LL_RTC_ALMB_GetSubSecondMask</i></a> <a href="#"><i>LL_RTC_ALMB_SetSubSecondMask</i></a>
	SS	<a href="#"><i>LL_RTC_ALMB_SetSubSecond</i></a>
BKPxR	BKP	<a href="#"><i>LL_RTC_BAK_GetRegister</i></a> <a href="#"><i>LL_RTC_BAK_SetRegister</i></a>
	CALM	<a href="#"><i>LL_RTC_CAL_SetMinus</i></a>
CALR	CALP	<a href="#"><i>LL_RTC_CAL_IsPulseInserted</i></a> <a href="#"><i>LL_RTC_CAL_SetPulse</i></a>
	CALW16	<a href="#"><i>LL_RTC_CAL_SetPeriod</i></a>
		<a href="#"><i>LL_RTC_CAL_SetPeriod</i></a>

Register	Field	Function
CR	CALW8	<i>LL_RTC_CAL_GetPeriod</i>
		<i>LL_RTC_CAL_SetPeriod</i>
	ADD1H	<i>LL_RTC_TIME_IncHour</i>
		<i>LL_RTC_ALMA_Disable</i>
		<i>LL_RTC_ALMA_Enable</i>
	ALRAE	<i>LL_RTC_DisableIT_ALRA</i>
		<i>LL_RTC_EnableIT_ALRA</i>
		<i>LL_RTC_IsEnabledIT_ALRA</i>
	ALRAIE	<i>LL_RTC_ALMB_Disable</i>
		<i>LL_RTC_ALMB_Enable</i>
	ALRBE	<i>LL_RTC_DisableIT_ALRB</i>
		<i>LL_RTC_EnableIT_ALRB</i>
		<i>LL_RTC_IsEnabledIT_ALRB</i>
	BCK	<i>LL_RTC_TIME_DisableDayLightStore</i>
		<i>LL_RTC_TIME_EnableDayLightStore</i>
		<i>LL_RTC_TIME_IsDayLightStoreEnabled</i>
	BYPSHAD	<i>LL_RTC_DisableShadowRegBypass</i>
		<i>LL_RTC_EnableShadowRegBypass</i>
		<i>LL_RTC_IsShadowRegBypassEnabled</i>
	COE	<i>LL_RTC_CAL_GetOutputFreq</i>
		<i>LL_RTC_CAL_SetOutputFreq</i>
	COSEL	<i>LL_RTC_CAL_GetOutputFreq</i>
		<i>LL_RTC_CAL_SetOutputFreq</i>
	FMT	<i>LL_RTC_GetHourFormat</i>
		<i>LL_RTC_SetHourFormat</i>
	OSEL	<i>LL_RTC_GetAlarmOutEvent</i>
		<i>LL_RTC_SetAlarmOutEvent</i>
	POL	<i>LL_RTC_GetOutputPolarity</i>
		<i>LL_RTC_SetOutputPolarity</i>
	REFCKON	<i>LL_RTC_DisableRefClock</i>
		<i>LL_RTC_EnableRefClock</i>
	SUB1H	<i>LL_RTC_TIME_DecHour</i>
	TSE	<i>LL_RTC_TS_Disable</i>
		<i>LL_RTC_TS_Enable</i>
	TSEDGE	<i>LL_RTC_TS_GetActiveEdge</i>
		<i>LL_RTC_TS_SetActiveEdge</i>

Register	Field	Function
DR	TSIE	<i>LL_RTC_DisableIT_TS</i>
		<i>LL_RTC_EnableIT_TS</i>
		<i>LL_RTC_IsEnabledIT_TS</i>
	WUCKSEL	<i>LL_RTC_WAKEUP_GetClock</i>
		<i>LL_RTC_WAKEUP_SetClock</i>
	WUTE	<i>LL_RTC_WAKEUP_Disable</i>
		<i>LL_RTC_WAKEUP_Enable</i>
		<i>LL_RTC_WAKEUP_IsEnabled</i>
	WUTIE	<i>LL_RTC_DisableIT_WUT</i>
		<i>LL_RTC_EnableIT_WUT</i>
		<i>LL_RTC_IsEnabledIT_WUT</i>
	DT	<i>LL_RTC_DATE_Config</i>
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetDay</i>
		<i>LL_RTC_DATE_SetDay</i>
	DU	<i>LL_RTC_DATE_Config</i>
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetDay</i>
		<i>LL_RTC_DATE_SetDay</i>
	MT	<i>LL_RTC_DATE_Config</i>
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetMonth</i>
		<i>LL_RTC_DATE_SetMonth</i>
	MU	<i>LL_RTC_DATE_Config</i>
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetMonth</i>
		<i>LL_RTC_DATE_SetMonth</i>
	WDU	<i>LL_RTC_DATE_Config</i>
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetWeekDay</i>
		<i>LL_RTC_DATE_SetWeekDay</i>
	YT	<i>LL_RTC_DATE_Config</i>
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetYear</i>
		<i>LL_RTC_DATE_SetYear</i>
	YU	<i>LL_RTC_DATE_Config</i>

Register	Field	Function
		<i>LL_RTC_DATE_Get</i>
		<i>LL_RTC_DATE_GetYear</i>
		<i>LL_RTC_DATE_SetYear</i>
	ALRAF	<i>LL_RTC_ClearFlag_ALRA</i> <i>LL_RTC_IsActiveFlag_ALRA</i>
	ALRAWF	<i>LL_RTC_IsActiveFlag_ALRAW</i>
	ALRBF	<i>LL_RTC_ClearFlag_ALRB</i> <i>LL_RTC_IsActiveFlag_ALRB</i>
	ALRBWF	<i>LL_RTC_IsActiveFlag_ALRBW</i>
	INIT	<i>LL_RTC_DisableInitMode</i> <i>LL_RTC_EnableInitMode</i>
	INITF	<i>LL_RTC_IsActiveFlag_INIT</i>
	INITS	<i>LL_RTC_IsActiveFlag_INITS</i>
	RECALPF	<i>LL_RTC_IsActiveFlag_RECALP</i>
	RSF	<i>LL_RTC_ClearFlag_RS</i> <i>LL_RTC_IsActiveFlag_RS</i>
ISR	SHPF	<i>LL_RTC_IsActiveFlag_SHP</i>
	TAMP1F	<i>LL_RTC_ClearFlag_TAMP1</i> <i>LL_RTC_IsActiveFlag_TAMP1</i>
	TAMP2F	<i>LL_RTC_ClearFlag_TAMP2</i> <i>LL_RTC_IsActiveFlag_TAMP2</i>
	TAMP3F	<i>LL_RTC_ClearFlag_TAMP3</i> <i>LL_RTC_IsActiveFlag_TAMP3</i>
	TSF	<i>LL_RTC_ClearFlag_TS</i> <i>LL_RTC_IsActiveFlag_TS</i>
	TSOVF	<i>LL_RTC_ClearFlag_TSOV</i> <i>LL_RTC_IsActiveFlag_TSOV</i>
	WUTF	<i>LL_RTC_ClearFlag_WUT</i> <i>LL_RTC_IsActiveFlag_WUT</i>
	WUTWF	<i>LL_RTC_IsActiveFlag_WUTW</i>
OR	ALARMOUTTYPE	<i>LL_RTC_GetAlarmOutputType</i> <i>LL_RTC_SetAlarmOutputType</i>
	OUT_RMP	<i>LL_RTC_DisableOutRemap</i> <i>LL_RTC_EnableOutRemap</i>
PRER	PREDIV_A	<i>LL_RTC_GetAsynchPrescaler</i> <i>LL_RTC_SetAsynchPrescaler</i>

Register	Field	Function	
	PREDIV_S	<a href="#">LL_RTC_GetSynchPrescaler</a> <a href="#">LL_RTC_SetSynchPrescaler</a>	
SHIFTR	ADD1S	<a href="#">LL_RTC_TIME_Synchronize</a>	
	SUBFS	<a href="#">LL_RTC_TIME_Synchronize</a>	
SSR	SS	<a href="#">LL_RTC_TIME_GetSubSecond</a>	
TAMPCR	TAMP1E	<a href="#">LL_RTC_TAMPER_Disable</a> <a href="#">LL_RTC_TAMPER_Enable</a>	
		TAMP1IE	<a href="#">LL_RTC_DisableIT_TAMP1</a> <a href="#">LL_RTC_EnableIT_TAMP1</a> <a href="#">LL_RTC_IsEnabledIT_TAMP1</a>
	TAMP1MF		<a href="#">LL_RTC_TAMPER_DisableMask</a> <a href="#">LL_RTC_TAMPER_EnableMask</a>
			TAMP1NOERASE
	TAMP1TRG	<a href="#">LL_RTC_TAMPER_DisableActiveLevel</a> <a href="#">LL_RTC_TAMPER_EnableActiveLevel</a>	
		TAMP2E	<a href="#">LL_RTC_TAMPER_Disable</a> <a href="#">LL_RTC_TAMPER_Enable</a>
	TAMP2IE		<a href="#">LL_RTC_DisableIT_TAMP2</a> <a href="#">LL_RTC_EnableIT_TAMP2</a> <a href="#">LL_RTC_IsEnabledIT_TAMP2</a>
		TAMP2MF	<a href="#">LL_RTC_TAMPER_DisableMask</a> <a href="#">LL_RTC_TAMPER_EnableMask</a>
			TAMP2NOERASE
	TAMP2TRG	<a href="#">LL_RTC_TAMPER_DisableActiveLevel</a> <a href="#">LL_RTC_TAMPER_EnableActiveLevel</a>	
		TAMP3E	<a href="#">LL_RTC_TAMPER_Disable</a> <a href="#">LL_RTC_TAMPER_Enable</a>
	TAMP3IE		<a href="#">LL_RTC_DisableIT_TAMP3</a> <a href="#">LL_RTC_EnableIT_TAMP3</a> <a href="#">LL_RTC_IsEnabledIT_TAMP3</a>
		TAMP3MF	<a href="#">LL_RTC_TAMPER_DisableMask</a> <a href="#">LL_RTC_TAMPER_EnableMask</a>
			TAMP3NOERASE

Register	Field	Function
	TAMP3TRG	<i>LL_RTC_TAMPER_DisableActiveLevel</i>
		<i>LL_RTC_TAMPER_EnableActiveLevel</i>
	TAMPFLT	<i>LL_RTC_TAMPER_GetFilterCount</i>
		<i>LL_RTC_TAMPER_SetFilterCount</i>
	TAMPFREQ	<i>LL_RTC_TAMPER_GetSamplingFreq</i>
		<i>LL_RTC_TAMPER_SetSamplingFreq</i>
	TAMPIE	<i>LL_RTC_DisableIT_TAMP</i>
		<i>LL_RTC_EnableIT_TAMP</i>
		<i>LL_RTC_IsEnabledIT_TAMP</i>
	TAMPPRCH	<i>LL_RTC_TAMPER_GetPrecharge</i>
		<i>LL_RTC_TAMPER_SetPrecharge</i>
	TAMPPUDIS	<i>LL_RTC_TAMPER_DisablePullUp</i>
		<i>LL_RTC_TAMPER_EnablePullUp</i>
	TAMPTS	<i>LL_RTC_TS_DisableOnTamper</i>
		<i>LL_RTC_TS_EnableOnTamper</i>
TR	HT	<i>LL_RTC_TIME_Config</i>
		<i>LL_RTC_TIME_Get</i>
		<i>LL_RTC_TIME_GetHour</i>
		<i>LL_RTC_TIME_SetHour</i>
	HU	<i>LL_RTC_TIME_Config</i>
		<i>LL_RTC_TIME_Get</i>
		<i>LL_RTC_TIME_GetHour</i>
		<i>LL_RTC_TIME_SetHour</i>
	MNT	<i>LL_RTC_TIME_Config</i>
		<i>LL_RTC_TIME_Get</i>
		<i>LL_RTC_TIME_GetMinute</i>
		<i>LL_RTC_TIME_SetMinute</i>
	MNU	<i>LL_RTC_TIME_Config</i>
		<i>LL_RTC_TIME_Get</i>
		<i>LL_RTC_TIME_GetMinute</i>
		<i>LL_RTC_TIME_SetMinute</i>
	PM	<i>LL_RTC_TIME_Config</i>
		<i>LL_RTC_TIME_GetFormat</i>
		<i>LL_RTC_TIME_SetFormat</i>
	ST	<i>LL_RTC_TIME_Config</i>
		<i>LL_RTC_TIME_Get</i>

Register	Field	Function
TSDR	SU	<a href="#"><i>LL_RTC_TIME_GetSecond</i></a>
		<a href="#"><i>LL_RTC_TIME_SetSecond</i></a>
	DT	<a href="#"><i>LL_RTC_TIME_Config</i></a>
		<a href="#"><i>LL_RTC_TIME_Get</i></a>
		<a href="#"><i>LL_RTC_TIME_GetSecond</i></a>
	DU	<a href="#"><i>LL_RTC_TIME_SetSecond</i></a>
		<a href="#"><i>LL_RTC_TS_GetDate</i></a>
		<a href="#"><i>LL_RTC_TS_GetDay</i></a>
		<a href="#"><i>LL_RTC_TS_GetDate</i></a>
		<a href="#"><i>LL_RTC_TS_GetDay</i></a>
	MT	<a href="#"><i>LL_RTC_TS_GetDate</i></a>
		<a href="#"><i>LL_RTC_TS_GetMonth</i></a>
	MU	<a href="#"><i>LL_RTC_TS_GetDate</i></a>
		<a href="#"><i>LL_RTC_TS_GetMonth</i></a>
	WUDU	<a href="#"><i>LL_RTC_TS_GetDate</i></a>
		<a href="#"><i>LL_RTC_TS_GetWeekDay</i></a>
TSSSR	SS	<a href="#"><i>LL_RTC_TS_GetSubSecond</i></a>
TSTR	HT	<a href="#"><i>LL_RTC_TS_GetHour</i></a>
		<a href="#"><i>LL_RTC_TS_GetTime</i></a>
	HU	<a href="#"><i>LL_RTC_TS_GetHour</i></a>
		<a href="#"><i>LL_RTC_TS_GetTime</i></a>
	MNT	<a href="#"><i>LL_RTC_TS_GetMinute</i></a>
		<a href="#"><i>LL_RTC_TS_GetTime</i></a>
	MNU	<a href="#"><i>LL_RTC_TS_GetMinute</i></a>
		<a href="#"><i>LL_RTC_TS_GetTime</i></a>
	PM	<a href="#"><i>LL_RTC_TS_GetTimeFormat</i></a>
	ST	<a href="#"><i>LL_RTC_TS_GetSecond</i></a>
		<a href="#"><i>LL_RTC_TS_GetTime</i></a>
	SU	<a href="#"><i>LL_RTC_TS_GetSecond</i></a>
		<a href="#"><i>LL_RTC_TS_GetTime</i></a>
WPR	KEY	<a href="#"><i>LL_RTC_DisableWriteProtection</i></a>
		<a href="#"><i>LL_RTC_EnableWriteProtection</i></a>
WUTR	WUT	<a href="#"><i>LL_RTC_WAKEUP_GetAutoReload</i></a>
		<a href="#"><i>LL_RTC_WAKEUP_SetAutoReload</i></a>

## 78.20 SPI

Table 44: Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CR1	BIDIMODE	<i>LL_SPI_GetTransferDirection</i>
		<i>LL_SPI_SetTransferDirection</i>
	BIDIOE	<i>LL_SPI_GetTransferDirection</i>
		<i>LL_SPI_SetTransferDirection</i>
	BR	<i>LL_SPI_GetBaudRatePrescaler</i>
		<i>LL_SPI_SetBaudRatePrescaler</i>
	CPHA	<i>LL_SPI_GetClockPhase</i>
		<i>LL_SPI_SetClockPhase</i>
	CPOL	<i>LL_SPI_GetClockPolarity</i>
		<i>LL_SPI_SetClockPolarity</i>
	CRCEN	<i>LL_SPI_DisableCRC</i>
		<i>LL_SPI_EnableCRC</i>
		<i>LL_SPI_IsEnabledCRC</i>
	CRCNEXT	<i>LL_SPI_SetCRCNext</i>
	DFF	<i>LL_SPI_GetDataWidth</i>
		<i>LL_SPI_SetDataWidth</i>
	LSBFIRST	<i>LL_SPI_GetTransferBitOrder</i>
		<i>LL_SPI_SetTransferBitOrder</i>
	MSTR	<i>LL_SPI_GetMode</i>
		<i>LL_SPI_SetMode</i>
	RXONLY	<i>LL_SPI_GetTransferDirection</i>
		<i>LL_SPI_SetTransferDirection</i>
	SPE	<i>LL_SPI_Disable</i>
		<i>LL_SPI_Enable</i>
		<i>LL_SPI_IsEnabled</i>
	SSI	<i>LL_SPI_GetMode</i>
		<i>LL_SPI_SetMode</i>
	SSM	<i>LL_SPI_GetNSSMode</i>
		<i>LL_SPI_SetNSSMode</i>
CR2	ERRIE	<i>LL_SPI_DisableIT_ERR</i>
		<i>LL_SPI_EnableIT_ERR</i>
		<i>LL_SPI_IsEnabledIT_ERR</i>
	FRF	<i>LL_SPI_GetStandard</i>
		<i>LL_SPI_SetStandard</i>

Register	Field	Function
	RXDMAEN	<i>LL_SPI_DisableDMAReq_RX</i>
		<i>LL_SPI_EnableDMAReq_RX</i>
		<i>LL_SPI_IsEnabledDMAReq_RX</i>
	RXNEIE	<i>LL_SPI_DisableIT_RXNE</i>
		<i>LL_SPI_EnableIT_RXNE</i>
		<i>LL_SPI_IsEnabledIT_RXNE</i>
	SSOE	<i>LL_SPI_GetNSSMode</i>
		<i>LL_SPI_SetNSSMode</i>
	TXDMAEN	<i>LL_SPI_DisableDMAReq_TX</i>
		<i>LL_SPI_EnableDMAReq_TX</i>
		<i>LL_SPI_IsEnabledDMAReq_TX</i>
	TXEIE	<i>LL_SPI_DisableIT_TXE</i>
		<i>LL_SPI_EnableIT_TXE</i>
		<i>LL_SPI_IsEnabledIT_TXE</i>
CRCPR	CRCPOLY	<i>LL_SPI_GetCRCPolynomial</i>
		<i>LL_SPI_SetCRCPolynomial</i>
DR	DR	<i>LL_SPI_DMA_GetRegAddr</i>
		<i>LL_SPI_ReceiveData16</i>
		<i>LL_SPI_ReceiveData8</i>
		<i>LL_SPI_TransmitData16</i>
		<i>LL_SPI_TransmitData8</i>
RXCRCR	RXCRC	<i>LL_SPI_GetRxCRC</i>
SR	BSY	<i>LL_SPI_IsActiveFlag_BSY</i>
	CRCERR	<i>LL_SPI_ClearFlag_CRCERR</i>
		<i>LL_SPI_IsActiveFlag_CRCERR</i>
	FRE	<i>LL_SPI_ClearFlag_FRE</i>
		<i>LL_SPI_IsActiveFlag_FRE</i>
	MODF	<i>LL_SPI_ClearFlag_MODF</i>
		<i>LL_SPI_IsActiveFlag_MODF</i>
	OVR	<i>LL_SPI_ClearFlag_OVR</i>
		<i>LL_SPI_IsActiveFlag_OVR</i>
	RXNE	<i>LL_SPI_IsActiveFlag_RXNE</i>
	TXE	<i>LL_SPI_IsActiveFlag_TXE</i>
TXCRCR	TXCRC	<i>LL_SPI_GetTxCRC</i>

## 78.21 SYSTEM

Table 45: Correspondence between SYSTEM registers and SYSTEM low-layer driver functions

Register	Field	Function
APB1FZ	DBG_I2C1_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_I2C2_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_I2C3_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_IWDG_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_LPTIMER_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_RTC_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_TIM2_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_TIM3_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_TIM6_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_TIM7_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
	DBG_WWDG_STOP	<a href="#"><i>LL_DBGMCU_ABP1_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP1_GRP1_UnFreezePeriph</i></a>
APB2FZ	DBG_TIM21_STOP	<a href="#"><i>LL_DBGMCU_ABP2_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP2_GRP1_UnFreezePeriph</i></a>
	DBG_TIM22_STOP	<a href="#"><i>LL_DBGMCU_ABP2_GRP1_FreezePeriph</i></a>
		<a href="#"><i>LL_DBGMCU_ABP2_GRP1_UnFreezePeriph</i></a>
DBGMCU_CR	DBG_SLEEP	<a href="#"><i>LL_DBGMCU_DisableDBGSleepMode</i></a>
		<a href="#"><i>LL_DBGMCU_EnableDBGSleepMode</i></a>
	DBG_STANDBY	<a href="#"><i>LL_DBGMCU_DisableDBGStandbyMode</i></a>
		<a href="#"><i>LL_DBGMCU_EnableDBGStandbyMode</i></a>
	DBG_STOP	<a href="#"><i>LL_DBGMCU_DisableDBGStopMode</i></a>
		<a href="#"><i>LL_DBGMCU_EnableDBGStopMode</i></a>
DBGMCU_IDCODE	DEV_ID	<a href="#"><i>LL_DBGMCU_GetDeviceID</i></a>
	REV_ID	<a href="#"><i>LL_DBGMCU_GetRevisionID</i></a>

Register	Field	Function
FLASH_ACR	DISAB_BUF	<code>LL_FLASH_DisableBuffers</code>
		<code>LL_FLASH_EnableBuffers</code>
	LATENCY	<code>LL_FLASH_GetLatency</code>
		<code>LL_FLASH_SetLatency</code>
	PRE_READ	<code>LL_FLASH_DisablePreRead</code>
		<code>LL_FLASH_EnablePreRead</code>
	PRFTEN	<code>LL_FLASH_DisablePrefetch</code>
		<code>LL_FLASH_EnablePrefetch</code>
		<code>LL_FLASH_IsPrefetchEnabled</code>
	RUN_PD	<code>LL_FLASH_DisableRunPowerDown</code>
		<code>LL_FLASH_EnableRunPowerDown</code>
	SLEEP_PD	<code>LL_FLASH_DisableSleepPowerDown</code>
		<code>LL_FLASH_EnableSleepPowerDown</code>
FLASH_PDKEYR	PDKEY1	<code>LL_FLASH_DisableRunPowerDown</code>
		<code>LL_FLASH_EnableRunPowerDown</code>
	PDKEY2	<code>LL_FLASH_DisableRunPowerDown</code>
		<code>LL_FLASH_EnableRunPowerDown</code>
SYSCFG_CFGR1	BOOT_MODE	<code>LL_SYSCFG_GetBootMode</code>
	MEM_MODE	<code>LL_SYSCFG_GetRemapMemory</code>
		<code>LL_SYSCFG_SetRemapMemory</code>
	UFB	<code>LL_SYSCFG_GetFlashBankMode</code>
		<code>LL_SYSCFG_SetFlashBankMode</code>
SYSCFG_CFGR2	CAPA	<code>LL_SYSCFG_GetVLCDRailConnection</code>
		<code>LL_SYSCFG_SetVLCDRailConnection</code>
	FWDIS	<code>LL_SYSCFG_EnableFirewall</code>
		<code>LL_SYSCFG_IsEnabledFirewall</code>
	I2C_PBx_FMP	<code>LL_SYSCFG_DisableFastModePlus</code>
		<code>LL_SYSCFG_EnableFastModePlus</code>
	I2Cx_FMP	<code>LL_SYSCFG_DisableFastModePlus</code>
		<code>LL_SYSCFG_EnableFastModePlus</code>
SYSCFG_CFGR3	ENBUF_SENSOR_ADC	<code>LL_SYSCFG_TEMPSENSOR_Disable</code>
		<code>LL_SYSCFG_TEMPSENSOR_Enable</code>
	ENBUF_VREFINT_ADC	<code>LL_SYSCFG_VREFINT_DisableADC</code>
		<code>LL_SYSCFG_VREFINT_EnableADC</code>
	ENBUF_VREFINT_COMP	<code>LL_SYSCFG_VREFINT_DisableCOMP</code>
		<code>LL_SYSCFG_VREFINT_EnableCOMP</code>

Register	Field	Function
	ENREF_HSI48	<code>LL_SYSCFG_VREFINT_DisableHSI48</code>
		<code>LL_SYSCFG_VREFINT_EnableHSI48</code>
	REF_LOCK	<code>LL_SYSCFG_VREFINT_IsLocked</code>
		<code>LL_SYSCFG_VREFINT_Lock</code>
	SEL_VREF_OUT	<code>LL_SYSCFG_VREFINT_GetConnection</code>
		<code>LL_SYSCFG_VREFINT_SetConnection</code>
	VREFINT_RDYF	<code>LL_SYSCFG_VREFINT_IsReady</code>
SYSCFG_EXTICR1	EXTI0	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI1	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI2	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI3	<code>LL_SYSCFG_SetEXTISource</code>
SYSCFG_EXTICR2	EXTI4	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI5	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI6	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI7	<code>LL_SYSCFG_SetEXTISource</code>
SYSCFG_EXTICR3	EXTI10	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI11	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI18	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI9	<code>LL_SYSCFG_SetEXTISource</code>
SYSCFG_EXTICR4	EXTI12	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI13	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI14	<code>LL_SYSCFG_SetEXTISource</code>
	EXTI15	<code>LL_SYSCFG_SetEXTISource</code>

## 78.22 TIM

Table 46: Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
ARR	ARR	<code>LL_TIM_SetAutoReload</code>
		<code>LL_TIM_SetAutoReload</code>
CCER	CC1E	<code>LL_TIM_CC_DisableChannel</code>
		<code>LL_TIM_CC_EnableChannel</code>
		<code>LL_TIM_CC_IsEnabledChannel</code>
	CC1NP	<code>LL_TIM_IC_Config</code>
		<code>LL_TIM_IC_GetPolarity</code>
		<code>LL_TIM_IC_SetPolarity</code>
	CC1P	<code>LL_TIM_IC_Config</code>

Register	Field	Function
		<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
		<i>LL_TIM_OC_ConfigOutput</i>
		<i>LL_TIM_OC_GetPolarity</i>
		<i>LL_TIM_OC_SetPolarity</i>
	CC2E	<i>LL_TIM_CC_DisableChannel</i>
		<i>LL_TIM_CC_EnableChannel</i>
		<i>LL_TIM_CC_IsEnabledChannel</i>
	CC2NP	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
	CC2P	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
		<i>LL_TIM_OC_ConfigOutput</i>
		<i>LL_TIM_OC_GetPolarity</i>
		<i>LL_TIM_OC_SetPolarity</i>
	CC3E	<i>LL_TIM_CC_DisableChannel</i>
		<i>LL_TIM_CC_EnableChannel</i>
		<i>LL_TIM_CC_IsEnabledChannel</i>
	CC3NP	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
	CC3P	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
		<i>LL_TIM_OC_ConfigOutput</i>
		<i>LL_TIM_OC_GetPolarity</i>
		<i>LL_TIM_OC_SetPolarity</i>
	CC4E	<i>LL_TIM_CC_DisableChannel</i>
		<i>LL_TIM_CC_EnableChannel</i>
		<i>LL_TIM_CC_IsEnabledChannel</i>
	CC4NP	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
	CC4P	<i>LL_TIM_IC_Config</i>

Register	Field	Function
CCMR1	CC1S	<i>LL_TIM_IC_GetPolarity</i>
		<i>LL_TIM_IC_SetPolarity</i>
		<i>LL_TIM_OC_ConfigOutput</i>
		<i>LL_TIM_OC_GetPolarity</i>
		<i>LL_TIM_OC_SetPolarity</i>
	CC2S	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetActiveInput</i>
		<i>LL_TIM_IC_SetActiveInput</i>
		<i>LL_TIM_OC_ConfigOutput</i>
	IC1F	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetFilter</i>
		<i>LL_TIM_IC_SetFilter</i>
	IC1PSC	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPrescaler</i>
		<i>LL_TIM_IC_SetPrescaler</i>
	IC2F	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetFilter</i>
		<i>LL_TIM_IC_SetFilter</i>
	IC2PSC	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPrescaler</i>
		<i>LL_TIM_IC_SetPrescaler</i>
	OC1CE	<i>LL_TIM_OC_DisableClear</i>
		<i>LL_TIM_OC_EnableClear</i>
		<i>LL_TIM_OC_IsEnabledClear</i>
	OC1FE	<i>LL_TIM_OC_DisableFast</i>
		<i>LL_TIM_OC_EnableFast</i>
		<i>LL_TIM_OC_IsEnabledFast</i>
	OC1M	<i>LL_TIM_OC_GetMode</i>
		<i>LL_TIM_OC_SetMode</i>
	OC1PE	<i>LL_TIM_OC_DisablePreload</i>
		<i>LL_TIM_OC_EnablePreload</i>
		<i>LL_TIM_OC_IsEnabledPreload</i>

Register	Field	Function
CCMR2	OC2CE	<i>LL_TIM_OC_DisableClear</i>
		<i>LL_TIM_OC_EnableClear</i>
		<i>LL_TIM_OC_IsEnabledClear</i>
	OC2FE	<i>LL_TIM_OC_DisableFast</i>
		<i>LL_TIM_OC_EnableFast</i>
		<i>LL_TIM_OC_IsEnabledFast</i>
	OC2M	<i>LL_TIM_OC_GetMode</i>
		<i>LL_TIM_OC_SetMode</i>
	OC2PE	<i>LL_TIM_OC_DisablePreload</i>
		<i>LL_TIM_OC_EnablePreload</i>
		<i>LL_TIM_OC_IsEnabledPreload</i>
	CC3S	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetActiveInput</i>
		<i>LL_TIM_IC_SetActiveInput</i>
		<i>LL_TIM_OC_ConfigOutput</i>
	CC4S	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetActiveInput</i>
		<i>LL_TIM_IC_SetActiveInput</i>
		<i>LL_TIM_OC_ConfigOutput</i>
	IC3F	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetFilter</i>
		<i>LL_TIM_IC_SetFilter</i>
	IC3PSC	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPrescaler</i>
		<i>LL_TIM_IC_SetPrescaler</i>
	IC4F	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetFilter</i>
		<i>LL_TIM_IC_SetFilter</i>
	IC4PSC	<i>LL_TIM_IC_Config</i>
		<i>LL_TIM_IC_GetPrescaler</i>
		<i>LL_TIM_IC_SetPrescaler</i>
	OC3CE	<i>LL_TIM_OC_DisableClear</i>
		<i>LL_TIM_OC_EnableClear</i>
		<i>LL_TIM_OC_IsEnabledClear</i>
	OC3FE	<i>LL_TIM_OC_DisableFast</i>
		<i>LL_TIM_OC_EnableFast</i>

Register	Field	Function
	OC3M	<i>LL_TIM_OC_IsEnabledFast</i>
		<i>LL_TIM_OC_GetMode</i>
		<i>LL_TIM_OC_SetMode</i>
	OC3PE	<i>LL_TIM_OC_DisablePreload</i>
		<i>LL_TIM_OC_EnablePreload</i>
		<i>LL_TIM_OC_IsEnabledPreload</i>
	OC4CE	<i>LL_TIM_OC_DisableClear</i>
		<i>LL_TIM_OC_EnableClear</i>
		<i>LL_TIM_OC_IsEnabledClear</i>
	OC4FE	<i>LL_TIM_OC_DisableFast</i>
		<i>LL_TIM_OC_EnableFast</i>
		<i>LL_TIM_OC_IsEnabledFast</i>
CCR1	CCR1	<i>LL_TIM_IC_GetCaptureCH1</i>
		<i>LL_TIM_OC_GetCompareCH1</i>
		<i>LL_TIM_OC_SetCompareCH1</i>
CCR2	CCR2	<i>LL_TIM_IC_GetCaptureCH2</i>
		<i>LL_TIM_OC_GetCompareCH2</i>
		<i>LL_TIM_OC_SetCompareCH2</i>
CCR3	CCR3	<i>LL_TIM_IC_GetCaptureCH3</i>
		<i>LL_TIM_OC_GetCompareCH3</i>
		<i>LL_TIM_OC_SetCompareCH3</i>
CCR4	CCR4	<i>LL_TIM_IC_GetCaptureCH4</i>
		<i>LL_TIM_OC_GetCompareCH4</i>
		<i>LL_TIM_OC_SetCompareCH4</i>
CNT	CNT	<i>LL_TIM_GetCounter</i>
		<i>LL_TIM_SetCounter</i>
CR1	ARPE	<i>LL_TIM_DisableARRPreload</i>
		<i>LL_TIM_EnableARRPreload</i>
		<i>LL_TIM_IsEnabledARRPreload</i>
	CEN	<i>LL_TIM_DisableCounter</i>
		<i>LL_TIM_EnableCounter</i>

Register	Field	Function
	CKD	<i>LL_TIM_IsEnabledCounter</i>
		<i>LL_TIM_GetClockDivision</i>
		<i>LL_TIM_SetClockDivision</i>
	CMS	<i>LL_TIM_GetCounterMode</i>
		<i>LL_TIM_SetCounterMode</i>
	DIR	<i>LL_TIM_GetCounterMode</i>
		<i>LL_TIM_GetDirection</i>
		<i>LL_TIM_SetCounterMode</i>
	OPM	<i>LL_TIM_GetOnePulseMode</i>
		<i>LL_TIM_SetOnePulseMode</i>
	UDIS	<i>LL_TIM_DisableUpdateEvent</i>
		<i>LL_TIM_EnableUpdateEvent</i>
		<i>LL_TIM_IsEnabledUpdateEvent</i>
	URS	<i>LL_TIM_GetUpdateSource</i>
		<i>LL_TIM_SetUpdateSource</i>
CR2	CCDS	<i>LL_TIM_CC_GetDMAReqTrigger</i>
		<i>LL_TIM_CC_SetDMAReqTrigger</i>
	MMS	<i>LL_TIM_SetTriggerOutput</i>
	TI1S	<i>LL_TIM_IC_DisableXORCombination</i>
		<i>LL_TIM_IC_EnableXORCombination</i>
		<i>LL_TIM_IC_IsEnabledXORCombination</i>
DCR	DBA	<i>LL_TIM_ConfigDMABurst</i>
	DBL	<i>LL_TIM_ConfigDMABurst</i>
DIER	CC1DE	<i>LL_TIM_DisableDMAReq_CC1</i>
		<i>LL_TIM_EnableDMAReq_CC1</i>
		<i>LL_TIM_IsEnabledDMAReq_CC1</i>
	CC1IE	<i>LL_TIM_DisableIT_CC1</i>
		<i>LL_TIM_EnableIT_CC1</i>
		<i>LL_TIM_IsEnabledIT_CC1</i>
	CC2DE	<i>LL_TIM_DisableDMAReq_CC2</i>
		<i>LL_TIM_EnableDMAReq_CC2</i>
		<i>LL_TIM_IsEnabledDMAReq_CC2</i>
	CC2IE	<i>LL_TIM_DisableIT_CC2</i>
		<i>LL_TIM_EnableIT_CC2</i>
		<i>LL_TIM_IsEnabledIT_CC2</i>
	CC3DE	<i>LL_TIM_DisableDMAReq_CC3</i>

Register	Field	Function
		<i>LL_TIM_EnableDMAReq_CC3</i>
		<i>LL_TIM_IsEnabledDMAReq_CC3</i>
	CC3IE	<i>LL_TIM_DisableIT_CC3</i>
		<i>LL_TIM_EnableIT_CC3</i>
		<i>LL_TIM_IsEnabledIT_CC3</i>
	CC4DE	<i>LL_TIM_DisableDMAReq_CC4</i>
		<i>LL_TIM_EnableDMAReq_CC4</i>
		<i>LL_TIM_IsEnabledDMAReq_CC4</i>
	CC4IE	<i>LL_TIM_DisableIT_CC4</i>
		<i>LL_TIM_EnableIT_CC4</i>
		<i>LL_TIM_IsEnabledIT_CC4</i>
	TDE	<i>LL_TIM_DisableDMAReq_TRIG</i>
		<i>LL_TIM_EnableDMAReq_TRIG</i>
		<i>LL_TIM_IsEnabledDMAReq_TRIG</i>
	TIE	<i>LL_TIM_DisableIT_TRIG</i>
		<i>LL_TIM_EnableIT_TRIG</i>
		<i>LL_TIM_IsEnabledIT_TRIG</i>
	UDE	<i>LL_TIM_DisableDMAReq_UPDATE</i>
		<i>LL_TIM_EnableDMAReq_UPDATE</i>
		<i>LL_TIM_IsEnabledDMAReq_UPDATE</i>
	UIE	<i>LL_TIM_DisableIT_UPDATE</i>
		<i>LL_TIM_EnableIT_UPDATE</i>
		<i>LL_TIM_IsEnabledIT_UPDATE</i>
EGR	CC1G	<i>LL_TIM_GenerateEvent_CC1</i>
	CC2G	<i>LL_TIM_GenerateEvent_CC2</i>
	CC3G	<i>LL_TIM_GenerateEvent_CC3</i>
	CC4G	<i>LL_TIM_GenerateEvent_CC4</i>
	TG	<i>LL_TIM_GenerateEvent_TRIG</i>
	UG	<i>LL_TIM_GenerateEvent_UPDATE</i>
PSC	PSC	<i>LL_TIM_GetPrescaler</i>
		<i>LL_TIM_SetPrescaler</i>
SMCR	ECE	<i>LL_TIM_DisableExternalClock</i>
		<i>LL_TIM_EnableExternalClock</i>
		<i>LL_TIM_IsEnabledExternalClock</i>
		<i>LL_TIM_SetClockSource</i>
	ETF	<i>LL_TIM_ConfigETR</i>

Register	Field	Function
SR	ETP	<i>LL_TIM_ConfigETR</i>
	ETPS	<i>LL_TIM_ConfigETR</i>
	MSM	<i>LL_TIM_DisableMasterSlaveMode</i>
		<i>LL_TIM_EnableMasterSlaveMode</i>
		<i>LL_TIM_IsEnabledMasterSlaveMode</i>
	OCCS	<i>LL_TIM_SetOCRefClearInputSource</i>
	SMS	<i>LL_TIM_SetClockSource</i>
		<i>LL_TIM_SetEncoderMode</i>
		<i>LL_TIM_SetSlaveMode</i>
	TS	<i>LL_TIM_SetTriggerInput</i>
TIMx_OR	CC1IF	<i>LL_TIM_ClearFlag_CC1</i>
		<i>LL_TIM_IsActiveFlag_CC1</i>
	CC1OF	<i>LL_TIM_ClearFlag_CC1OVR</i>
		<i>LL_TIM_IsActiveFlag_CC1OVR</i>
	CC2IF	<i>LL_TIM_ClearFlag_CC2</i>
		<i>LL_TIM_IsActiveFlag_CC2</i>
	CC2OF	<i>LL_TIM_ClearFlag_CC2OVR</i>
		<i>LL_TIM_IsActiveFlag_CC2OVR</i>
	CC3IF	<i>LL_TIM_ClearFlag_CC3</i>
		<i>LL_TIM_IsActiveFlag_CC3</i>
	CC3OF	<i>LL_TIM_ClearFlag_CC3OVR</i>
		<i>LL_TIM_IsActiveFlag_CC3OVR</i>
	CC4IF	<i>LL_TIM_ClearFlag_CC4</i>
		<i>LL_TIM_IsActiveFlag_CC4</i>
	CC4OF	<i>LL_TIM_ClearFlag_CC4OVR</i>
		<i>LL_TIM_IsActiveFlag_CC4OVR</i>
	TIF	<i>LL_TIM_ClearFlag_TRIG</i>
		<i>LL_TIM_IsActiveFlag_TRIG</i>
	UIF	<i>LL_TIM_ClearFlag_UPDATE</i>
		<i>LL_TIM_IsActiveFlag_UPDATE</i>
TIM21_OR	ETR_RMP	<i>LL_TIM_SetRemap</i>
	TI1_RMP	<i>LL_TIM_SetRemap</i>
	TI2_RMP	<i>LL_TIM_SetRemap</i>
TIM22_OR	ETR_RMP	<i>LL_TIM_SetRemap</i>
	TI1_RMP	<i>LL_TIM_SetRemap</i>
TIM2_OR	ETR_RMP	<i>LL_TIM_SetRemap</i>

Register	Field	Function
TIM3_OR	TI4_RMP	<i>LL_TIM_SetRemap</i>
	ETR_RMP	<i>LL_TIM_SetRemap</i>
	TI1_RMP	<i>LL_TIM_SetRemap</i>
	TI2_RMP	<i>LL_TIM_SetRemap</i>
	TI4_RMP	<i>LL_TIM_SetRemap</i>

## 78.23 USART

Table 47: Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
CR1	BRR	<i>LL_USART_GetBaudRate</i>
		<i>LL_USART_SetBaudRate</i>
	CMIE	<i>LL_USART_DisableIT_CM</i>
		<i>LL_USART_EnableIT_CM</i>
		<i>LL_USART_IsEnabledIT_CM</i>
	DEAT	<i>LL_USART_GetDEAssertionTime</i>
		<i>LL_USART_SetDEAssertionTime</i>
	DEDT	<i>LL_USART_GetDEDeassertionTime</i>
		<i>LL_USART_SetDEDeassertionTime</i>
	EOBIE	<i>LL_USART_DisableIT_EOB</i>
		<i>LL_USART_EnableIT_EOB</i>
		<i>LL_USART_IsEnabledIT_EOB</i>
	IDLEIE	<i>LL_USART_DisableIT_IDLE</i>
		<i>LL_USART_EnableIT_IDLE</i>
		<i>LL_USART_IsEnabledIT_IDLE</i>
	M0	<i>LL_USART_ConfigCharacter</i>
		<i>LL_USART_GetDataWidth</i>
		<i>LL_USART_SetDataWidth</i>
	M1	<i>LL_USART_ConfigCharacter</i>
		<i>LL_USART_GetDataWidth</i>
		<i>LL_USART_SetDataWidth</i>
	MME	<i>LL_USART_DisableMuteMode</i>
		<i>LL_USART_EnableMuteMode</i>
		<i>LL_USART_IsEnabledMuteMode</i>
	OVER8	<i>LL_USART_GetOverSampling</i>
		<i>LL_USART_SetOverSampling</i>
	PCE	<i>LL_USART_ConfigCharacter</i>

Register	Field	Function
		<code>LL_USART_GetParity</code>
		<code>LL_USART_SetParity</code>
	PEIE	<code>LL_USART_DisableIT_PE</code>
		<code>LL_USART_EnableIT_PE</code>
		<code>LL_USART_IsEnabledIT_PE</code>
	PS	<code>LL_USART_ConfigCharacter</code>
		<code>LL_USART_GetParity</code>
		<code>LL_USART_SetParity</code>
	RE	<code>LL_USART_DisableDirectionRx</code>
		<code>LL_USART_EnableDirectionRx</code>
		<code>LL_USART_GetTransferDirection</code>
		<code>LL_USART_SetTransferDirection</code>
	RTOIE	<code>LL_USART_DisableIT_RTO</code>
		<code>LL_USART_EnableIT_RTO</code>
		<code>LL_USART_IsEnabledIT_RTO</code>
	RXNEIE	<code>LL_USART_DisableIT_RXNE</code>
		<code>LL_USART_EnableIT_RXNE</code>
		<code>LL_USART_IsEnabledIT_RXNE</code>
	TCIE	<code>LL_USART_DisableIT_TC</code>
		<code>LL_USART_EnableIT_TC</code>
		<code>LL_USART_IsEnabledIT_TC</code>
	TE	<code>LL_USART_DisableDirectionTx</code>
		<code>LL_USART_EnableDirectionTx</code>
		<code>LL_USART_GetTransferDirection</code>
		<code>LL_USART_SetTransferDirection</code>
	TXEIE	<code>LL_USART_DisableIT_TXE</code>
		<code>LL_USART_EnableIT_TXE</code>
		<code>LL_USART_IsEnabledIT_TXE</code>
	UE	<code>LL_USART_Disable</code>
		<code>LL_USART_Enable</code>
		<code>LL_USART_IsEnabled</code>
	UESM	<code>LL_USART_DisableInStopMode</code>
		<code>LL_USART_EnableInStopMode</code>
		<code>LL_USART_IsEnabledInStopMode</code>
	WAKE	<code>LL_USART_GetWakeUpMethod</code>
		<code>LL_USART_SetWakeUpMethod</code>

Register	Field	Function
CR2	ABREN	<i>LL_USART_DisableAutoBaudRate</i>
		<i>LL_USART_EnableAutoBaudRate</i>
		<i>LL_USART_IsEnabledAutoBaud</i>
	ABRMODE	<i>LL_USART_GetAutoBaudRateMode</i>
		<i>LL_USART_SetAutoBaudRateMode</i>
	ADD	<i>LL_USART_ConfigNodeAddress</i>
		<i>LL_USART_GetNodeAddress</i>
	ADDM7	<i>LL_USART_ConfigNodeAddress</i>
		<i>LL_USART_GetNodeAddressLen</i>
	CLKEN	<i>LL_USART_ConfigAsyncMode</i>
		<i>LL_USART_ConfigHalfDuplexMode</i>
		<i>LL_USART_ConfigIrdaMode</i>
		<i>LL_USART_ConfigLINMode</i>
		<i>LL_USART_ConfigMultiProcessMode</i>
		<i>LL_USART_ConfigSmartcardMode</i>
		<i>LL_USART_ConfigSyncMode</i>
		<i>LL_USART_DisableSCLKOutput</i>
		<i>LL_USART_EnableSCLKOutput</i>
		<i>LL_USART_IsEnabledSCLKOutput</i>
	CPHA	<i>LL_USART_ConfigClock</i>
		<i>LL_USART_GetClockPhase</i>
		<i>LL_USART_SetClockPhase</i>
	CPOL	<i>LL_USART_ConfigClock</i>
		<i>LL_USART_GetClockPolarity</i>
		<i>LL_USART_SetClockPolarity</i>
	DATAINV	<i>LL_USART_GetBinaryDataLogic</i>
		<i>LL_USART_SetBinaryDataLogic</i>
	LBCL	<i>LL_USART_ConfigClock</i>
		<i>LL_USART_GetLastClkPulseOutput</i>
		<i>LL_USART_SetLastClkPulseOutput</i>
	LBDIE	<i>LL_USART_DisableIT_LBD</i>
		<i>LL_USART_EnableIT_LBD</i>
		<i>LL_USART_IsEnabledIT_LBD</i>
	LBDL	<i>LL_USART_GetLINBrkDetectionLen</i>
		<i>LL_USART_SetLINBrkDetectionLen</i>
	LINEN	<i>LL_USART_ConfigAsyncMode</i>

Register	Field	Function
		<i>LL_USART_ConfigHalfDuplexMode</i>
		<i>LL_USART_ConfigIrdaMode</i>
		<i>LL_USART_ConfigLINMode</i>
		<i>LL_USART_ConfigMultiProcessMode</i>
		<i>LL_USART_ConfigSmartcardMode</i>
		<i>LL_USART_ConfigSyncMode</i>
		<i>LL_USART_DisableLIN</i>
		<i>LL_USART_EnableLIN</i>
		<i>LL_USART_IsEnabledLIN</i>
MSBFIRST		<i>LL_USART_GetTransferBitOrder</i>
		<i>LL_USART_SetTransferBitOrder</i>
RTOEN		<i>LL_USART_DisableRxTimeout</i>
		<i>LL_USART_EnableRxTimeout</i>
		<i>LL_USART_IsEnabledRxTimeout</i>
RXINV		<i>LL_USART_GetRXPinLevel</i>
		<i>LL_USART_SetRXPinLevel</i>
STOP		<i>LL_USART_ConfigCharacter</i>
		<i>LL_USART_ConfigIrdaMode</i>
		<i>LL_USART_ConfigLINMode</i>
		<i>LL_USART_ConfigSmartcardMode</i>
		<i>LL_USART_GetStopBitsLength</i>
		<i>LL_USART_SetStopBitsLength</i>
SWAP		<i>LL_USART_GetTXRXSwap</i>
		<i>LL_USART_SetTXRXSwap</i>
TXINV		<i>LL_USART_GetTXPinLevel</i>
		<i>LL_USART_SetTXPinLevel</i>
CR3	CTSE	<i>LL_USART_DisableCTSHWFlowCtrl</i>
		<i>LL_USART_EnableCTSHWFlowCtrl</i>
		<i>LL_USART_GetHWFlowCtrl</i>
		<i>LL_USART_SetHWFlowCtrl</i>
	CTSIE	<i>LL_USART_DisableIT_CTS</i>
		<i>LL_USART_EnableIT_CTS</i>
		<i>LL_USART_IsEnabledIT_CTS</i>
	DDRE	<i>LL_USART_DisableDMAdeactOnRxErr</i>
		<i>LL_USART_EnableDMAdeactOnRxErr</i>
		<i>LL_USART_IsEnabledDMAdeactOnRxErr</i>

Register	Field	Function
	DEM	<code>LL_USART_DisableDEMode</code>
		<code>LL_USART_EnableDEMode</code>
		<code>LL_USART_IsEnabledDEMode</code>
	DEP	<code>LL_USART_GetDESignalPolarity</code>
		<code>LL_USART_SetDESignalPolarity</code>
	DMAR	<code>LL_USART_DisableDMAReq_RX</code>
		<code>LL_USART_EnableDMAReq_RX</code>
		<code>LL_USART_IsEnabledDMAReq_RX</code>
	DMAT	<code>LL_USART_DisableDMAReq_TX</code>
		<code>LL_USART_EnableDMAReq_TX</code>
		<code>LL_USART_IsEnabledDMAReq_TX</code>
	EIE	<code>LL_USART_DisableIT_ERROR</code>
		<code>LL_USART_EnableIT_ERROR</code>
		<code>LL_USART_IsEnabledIT_ERROR</code>
	HDSEL	<code>LL_USART_ConfigAsyncMode</code>
		<code>LL_USART_ConfigHalfDuplexMode</code>
		<code>LL_USART_ConfigIrdaMode</code>
		<code>LL_USART_ConfigLINMode</code>
		<code>LL_USART_ConfigMultiProcessMode</code>
		<code>LL_USART_ConfigSmartcardMode</code>
		<code>LL_USART_ConfigSyncMode</code>
		<code>LL_USART_DisableHalfDuplex</code>
		<code>LL_USART_EnableHalfDuplex</code>
		<code>LL_USART_IsEnabledHalfDuplex</code>
	IREN	<code>LL_USART_ConfigAsyncMode</code>
		<code>LL_USART_ConfigHalfDuplexMode</code>
		<code>LL_USART_ConfigIrdaMode</code>
		<code>LL_USART_ConfigLINMode</code>
		<code>LL_USART_ConfigMultiProcessMode</code>
		<code>LL_USART_ConfigSyncMode</code>
		<code>LL_USART_DisableIrda</code>
		<code>LL_USART_EnableIrda</code>
	IRLP	<code>LL_USART_GetIrdaPowerMode</code>
		<code>LL_USART_SetIrdaPowerMode</code>
	NACK	<code>LL_USART_DisableSmartcardNACK</code>

Register	Field	Function
		<i>LL_USART_EnableSmartcardNACK</i>
		<i>LL_USART_IsEnabledSmartcardNACK</i>
	ONEBIT	<i>LL_USART_DisableOneBitSamp</i>
		<i>LL_USART_EnableOneBitSamp</i>
		<i>LL_USART_IsEnabledOneBitSamp</i>
	OVRDIS	<i>LL_USART_DisableOverrunDetect</i>
		<i>LL_USART_EnableOverrunDetect</i>
		<i>LL_USART_IsEnabledOverrunDetect</i>
	RTSE	<i>LL_USART_DisableRTSHWFlowCtrl</i>
		<i>LL_USART_EnableRTSHWFlowCtrl</i>
		<i>LL_USART_GetHWFlowCtrl</i>
		<i>LL_USART_SetHWFlowCtrl</i>
	SCARCNT	<i>LL_USART_GetSmartcardAutoRetryCount</i>
		<i>LL_USART_SetSmartcardAutoRetryCount</i>
	SCEN	<i>LL_USART_ConfigAsyncMode</i>
		<i>LL_USART_ConfigHalfDuplexMode</i>
		<i>LL_USART_ConfigIrdaMode</i>
		<i>LL_USART_ConfigLINMode</i>
		<i>LL_USART_ConfigMultiProcessMode</i>
		<i>LL_USART_ConfigSmartcardMode</i>
		<i>LL_USART_ConfigSyncMode</i>
		<i>LL_USART_DisableSmartcard</i>
		<i>LL_USART_EnableSmartcard</i>
		<i>LL_USART_IsEnabledSmartcard</i>
	WUFIE	<i>LL_USART_DisableIT_WKUP</i>
		<i>LL_USART_EnableIT_WKUP</i>
		<i>LL_USART_IsEnabledIT_WKUP</i>
	WUS	<i>LL_USART_GetWKUPType</i>
		<i>LL_USART_SetWKUPType</i>
	GT	<i>LL_USART_GetSmartcardGuardTime</i>
		<i>LL_USART_SetSmartcardGuardTime</i>
	PSC	<i>LL_USART_GetIrdaPrescaler</i>
		<i>LL_USART_GetSmartcardPrescaler</i>
		<i>LL_USART_SetIrdaPrescaler</i>
		<i>LL_USART_SetSmartcardPrescaler</i>
ICR	CMCF	<i>LL_USART_ClearFlag_CM</i>

Register	Field	Function
	CTSCF	<a href="#">LL_USART_ClearFlag_nCTS</a>
	EOBCF	<a href="#">LL_USART_ClearFlag_EOB</a>
	FECF	<a href="#">LL_USART_ClearFlag_FE</a>
	IDLECF	<a href="#">LL_USART_ClearFlag_IDLE</a>
	LBDCF	<a href="#">LL_USART_ClearFlag_LBD</a>
	NCF	<a href="#">LL_USART_ClearFlag_NE</a>
	ORECF	<a href="#">LL_USART_ClearFlag_ORE</a>
	PECF	<a href="#">LL_USART_ClearFlag_PE</a>
	RTOCF	<a href="#">LL_USART_ClearFlag_RTO</a>
	TCCF	<a href="#">LL_USART_ClearFlag_TC</a>
	WUCF	<a href="#">LL_USART_ClearFlag_WKUP</a>
ISR	ABRE	<a href="#">LL_USART_IsActiveFlag_ABRE</a>
	ABRF	<a href="#">LL_USART_IsActiveFlag_ABR</a>
	BUSY	<a href="#">LL_USART_IsActiveFlag_BUSY</a>
	CMF	<a href="#">LL_USART_IsActiveFlag_CM</a>
	CTS	<a href="#">LL_USART_IsActiveFlag_CTS</a>
	CTSIF	<a href="#">LL_USART_IsActiveFlag_nCTS</a>
	EOBF	<a href="#">LL_USART_IsActiveFlag_EOB</a>
	FE	<a href="#">LL_USART_IsActiveFlag_FE</a>
	IDLE	<a href="#">LL_USART_IsActiveFlag_IDLE</a>
	LBDF	<a href="#">LL_USART_IsActiveFlag_LBD</a>
	NF	<a href="#">LL_USART_IsActiveFlag_NE</a>
	ORE	<a href="#">LL_USART_IsActiveFlag_ORE</a>
	PE	<a href="#">LL_USART_IsActiveFlag_PE</a>
	REACK	<a href="#">LL_USART_IsActiveFlag_REACK</a>
	RTOF	<a href="#">LL_USART_IsActiveFlag_RTO</a>
	RWU	<a href="#">LL_USART_IsActiveFlag_RWU</a>
	RXNE	<a href="#">LL_USART_IsActiveFlag_RXNE</a>
	SBKF	<a href="#">LL_USART_IsActiveFlag_SBK</a>
	TC	<a href="#">LL_USART_IsActiveFlag_TC</a>
	TEACK	<a href="#">LL_USART_IsActiveFlag_TEACK</a>
	TXE	<a href="#">LL_USART_IsActiveFlag_TXE</a>
	WUF	<a href="#">LL_USART_IsActiveFlag_WKUP</a>
RDR	RDR	<a href="#">LL_USART_DMA_GetRegAddr</a>
		<a href="#">LL_USART_ReceiveData8</a>
		<a href="#">LL_USART_ReceiveData9</a>

Register	Field	Function
RQR	ABRRQ	<i>LL_USART_RequestAutoBaudRate</i>
	MMRQ	<i>LL_USART_RequestEnterMuteMode</i>
	RXFRQ	<i>LL_USART_RequestRxDataFlush</i>
	SBKRQ	<i>LL_USART_RequestBreakSending</i>
	TXFRQ	<i>LL_USART_RequestTxDataFlush</i>
RTOR	BLEN	<i>LL_USART_GetBlockLength</i>
		<i>LL_USART_SetBlockLength</i>
	RTO	<i>LL_USART_GetRxTimeout</i>
		<i>LL_USART_SetRxTimeout</i>
TDR	TDR	<i>LL_USART_DMA_GetRegAddr</i>
		<i>LL_USART_TransmitData8</i>
		<i>LL_USART_TransmitData9</i>

## 78.24 WWDG

Table 48: Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	<i>LL_WWDG_EnableIT_EWKUP</i>
		<i>LL_WWDG_IsEnabledIT_EWKUP</i>
	W	<i>LL_WWDG_GetWindow</i>
		<i>LL_WWDG_SetWindow</i>
	WDGTB	<i>LL_WWDG_GetPrescaler</i>
		<i>LL_WWDG_SetPrescaler</i>
CR	T	<i>LL_WWDG_GetCounter</i>
		<i>LL_WWDG_SetCounter</i>
	WDGA	<i>LL_WWDG_Enable</i>
		<i>LL_WWDG_IsEnabled</i>
SR	EWIF	<i>LL_WWDG_ClearFlag_EWKUP</i>
		<i>LL_WWDG_IsActiveFlag_EWKUP</i>

## 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 STM32L0 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32L0 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 4.4: "Devices supported by HAL drivers"](#).

### What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

## Architecture

### How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32l0xx\_hal\_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration, Ethernet parameters configuration...)

A template is provided in the HAL drivers folders (stm32l0xx\_hal\_conf\_template.c).

### Which header files should I include in my application to use the HAL drivers?

Only stm32l0xx\_hal.h file has to be included.

### What is the difference between stm32l0xx\_hal\_ppp.c/h and stm32l0xx\_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 (stm32l0xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32l0xx\_hal\_ppp\_ex.c): It includes the specific APIs for specific device part number or family.

### Is it possible to use the APIs available in stm32l0xx\_ll\_ppp.c?

These APIs cannot be used directly because they are internal and offer services to upper layer drivers. As an example stm32l0xx\_ll\_fmc.c/h driver is used by stm32l0xx\_hal\_sram.c, stm32l0xx\_hal\_nor.c, stm32l0xx\_hal\_nand.c and stm32l0xx\_hal\_sdram.c drivers.

### Initialization and I/O operation functions

#### How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system\_stm32l0xx.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\_Delinit(), 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 stm32l0xx\_hal\_msp.c. A template is provided in the HAL driver folders (stm32l0xx\_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 **stm32l0xx\_hal\_msp.c**
6. Start your process operation by calling IO operation functions.

**What is the purpose of HAL\_PPP\_IRQHandler() function and when should I use it?**

HAL\_PPP\_IRQHandler() is used to handle interrupt process. It is called under PPP\_IRQHandler() function in stm32l0xx\_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 stm32l0xx\_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.

## 80 Revision history

Table 49: Document revision history

Date	Revision	Changes
03-Jun-2014	1	Initial release.
30-Jul-2015	2	<p><a href="#">Section 4.4: "Devices supported by HAL drivers"</a> added support for STM32L07xxx and STM32L08xxx and updated stm32l0xx_hal_dac_ex.c and stm32l0xx_hal_lcd.c support for STM32L05/6xx devices.</p> <p>Removed note related to STM32Cube V1.0 in <a href="#">Section 4.12.2.1: "HAL global initialization"</a>.</p> <p>Updated common macros in <a href="#">Section 4.9: "HAL common resources"</a></p> <p>Updated GPIO_TypeDef type (AF0) in <a href="#">Section 24: "HAL GPIO Generic Driver"</a>.</p> <p>Added new HAL_NVIC_GetPriority(IRQn_Type IRQn function in <a href="#">Section 12: "HAL CORTEX Generic Driver"</a>.</p>
20-Nov-2015	3	<p>Added stm32l0xx_flash.ld in <a href="#">Section 4.1.2: "User-application files"</a>.</p> <p>Added STM32L011xx, STM32L021xx, STM32L031xx and STM32L041xx in <a href="#">Section 4.4: "Devices supported by HAL drivers"</a>.</p> <p>Updated example of peripheral structure in <a href="#">Section 4.2.1: "Peripheral handle structures"</a>.</p> <p>Changed HAL_ADCEx_CalibrationStart() into HAL_ADCEx_Calibration_Start() in <a href="#">Table 10: "HAL extension APIs"</a>.</p> <p>Replaced GPIO_SPEED_LOW by GPIO_SPEED_FREQ_LOW, GPIO_SPEED_MEDIUM by GPIO_SPEED_FREQ_MEDIUM, GPIO_SPEED_FAST by GPIO_SPEED_FREQ_HIGH and GPIO_SPEED_HIGH by GPIO_SPEED_FREQ_VERY_HIGH speed in <a href="#">Table 12: "Description of GPIO_InitTypeDef structure"</a>.</p> <p>Updated example of system clock configuration code in <a href="#">Section 4.12.2.2: "System clock initialization"</a>.</p> <p>Add new API to control the MPU (see <a href="#">and</a> ).</p> <p>Renamed some HAL macros and driver define statements to ensure consistency all over STM32 families:</p> <ul style="list-style-type: none"> <li>• Added new define statements to specify the GPIO Speed (GPIO_SPEED_LOW replaced by GPIO_SPEED_FREQ_LOW for example).</li> <li>• Added new macro to check the Comparator inputs (IS_COMP1_LPTIMCONNECTION and IS_COMP2_LPTIMCONNECTION).</li> </ul>

Date	Revision	Changes
05-Apr-2016	4	<p>Updated .</p> <p>Updated <a href="#">Figure 7: "HAL driver model"</a>.</p> <p>Added <a href="#">Section 5: "Overview of Low Layer drivers"</a>.</p> <p>Added <a href="#">Section 6: "HAL and LL cohabitation"</a>.</p> <p><a href="#">Section 12.2: "CORTEX Firmware driver API description"</a>: remove __HAL_CORTEX_SYSTICKCLK_CONFIG(..) macro since this service is already covered by HAL_SYSTICK_CLKSourceConfig() function.</p> <p><a href="#">Section 19.3: "DMA Firmware driver defines"</a>: Added HAL_DMA_GET_COUNTER macro.</p> <p>Added descriptions of LL drivers for the following peripherals: ADC, COMP, CRC, CRS, DAC, DMA, EXTI, GPIO, I2C, I2S, IWDG, LPTIM, LPUART, PWR, RCC, RNG, RTC, SPI, TIM, USART, WWDG, as well as additional Low Level Bus, System, Cortex and Utilities APIs.</p>
07-Jun-2016	5	<p><b>HAL/LL COMP</b></p> <ul style="list-style-type: none"> <li>Added missing definition for COMP_INPUT_PLUS_IO6 and LL_COMP_INPUT_PLUS_IO6, supported by STM32L0 Category1 (STM32L011xx, STM32L021xx).</li> <li>Removed COMP_INVERTINGINPUT_IO3 definition.</li> <li>Renamed COMP_INVERTINGINPUT_IO2 to COMP_INPUT_MINUS_DAC1_CH2.</li> <li>The EXTI set-up is now managed by HAL_COMP_Init() function, using updated definitions of COMP_TRIGGERMODE_xxx.</li> </ul> <p>The functions HAL_COMP_Start_IT() and HAL_COMP_Stop_IT() have been removed.</p> <p><b>HAL LCD</b></p> <ul style="list-style-type: none"> <li>Corrected SYSCFG LCD External Capacitors definitions.</li> <li>Added new __HAL_SYSCFG_VLCD_CAPA_CONFIG() macro to configure the VLCD Decoupling capacitance connection.</li> <li>Added new __HAL_SYSCFG_GET_VLCD_CAPA_CONFIG() macro to return the decoupling of LCD capacitance configured by the user.</li> <li>Added LCD Voltage output buffer enable macro definitions.</li> </ul>

**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.

© 2016 STMicroelectronics – All rights reserved