RM0008
Reference manual

Low-, medium- and high-density STM32F101xx, STM32F102xx and STM32F103xx advanced ARM-based 32-bit MCUs

Introduction

This reference manual targets application developers. It provides complete information on how to use the low-, medium- and high-density STM32F101xx, STM32F102xx and STM32F103xx microcontroller memory and peripherals. The low-, medium- and high-density STM32F101xx, STM32F102xx and STM32F103xx will be referred to as STM32F10xxx throughout the document, unless otherwise specified.

The STM32F10xxx is a family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, mechanical and electrical device characteristics please refer to the low-, medium- and high-density *STM32F101xx and STM32F103xx* datasheets and to the low- and medium-density STM32F102xx datasheets.

For information on programming, erasing and protection of the internal Flash memory please refer to the *STM32F10xxx Flash programming manual.*

For information on the ARM Cortex™-M3 core, please refer to the *Cortex™-M3 Technical Reference Manual*.

Related documents

Available from *www.arm.com*:

- *Cortex™-M3 Technical Reference Manual, available from:*
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf

Available from *www.st.com*:

- *STM32F101xx STM32F103xx datasheets*
- *STM32F10xxx Flash programming manual*

Contents

1 Documentation conventions 32

1.1 List of abbreviations for registers 32

1.2 Glossary 32

1.3 Peripheral availability 32

2 Memory and bus architecture 33

2.1 System architecture 33

2.2 Memory organization 34

2.3 Memory map 35

2.3.1 Embedded SRAM 36

2.3.2 Bit banding 36

2.3.3 Embedded Flash memory 37

2.4 Boot configuration 41

3 CRC calculation unit 42

3.1 CRC introduction 42

3.2 CRC main features 42

3.3 CRC functional description 43

3.4 CRC registers 43

3.4.1 Data register (CRC_DR) 43

3.4.2 Independent data register (CRC_IDR) 44

3.4.3 Control register (CRC_CR) 44

3.4.4 CRC register map 44

4 Power control (PWR) 45

4.1 Power supplies 45

4.1.1 Independent A/D converter supply and reference voltage 46

4.1.2 Battery backup domain 46

4.1.3 Voltage regulator 47

4.2 Power supply supervisor 47

4.2.1 Power on reset (POR)/power down reset (PDR) 47

4.2.2 Programmable voltage detector (PVD) 48

4.3 Low-power modes 49

4.3.1 Slowing down system clocks 49
4.3.2 Peripheral clock gating 50
4.3.3 Sleep mode 50
4.3.4 Stop mode 51
4.3.5 Standby mode 52
4.3.6 Auto-wakeup (AWU) from low-power mode 54
4.4 Power control registers 54
4.4.1 Power control register (PWR_CR) 54
4.4.2 Power control/status register (PWR_CSR) 56
4.4.3 PWR register map 57

5 Backup registers (BKP) 58
5.1 BKP introduction 58
5.2 BKP main features 58
5.3 BKP functional description 59
5.3.1 Tamper detection 59
5.3.2 RTC calibration 59
5.4 BKP registers 60
5.4.1 Backup data register x (BKP_DRx) (x = 1 ..42) 60
5.4.2 RTC clock calibration register (BKP_RTCCR) 60
5.4.3 Backup control register (BKP_CR) 61
5.4.4 Backup control/status register (BKP_CSR) 62
5.4.5 BKP register map 63

6 Reset and clock control (RCC) 66
6.1 Reset 66
6.1.1 System reset 66
6.1.2 Power reset 67
6.1.3 Backup domain reset 67
6.2 Clocks 67
6.2.1 HSE clock 69
6.2.2 HSI clock 70
6.2.3 PLL 70
6.2.4 LSE clock 71
6.2.5 LSI clock 71
6.2.6 System clock (SYSCLK) selection 72

6.2.7 Clock security system (CSS) 72
6.2.8 RTC clock 72
6.2.9 Watchdog clock 73
6.2.10 Clock-out capability 73
6.3 RCC registers 74
6.3.1 Clock control register (RCC_CR) 74
6.3.2 Clock configuration register (RCC_CFGR) 75
6.3.3 Clock interrupt register (RCC_CIR) 78
6.3.4 APB2 peripheral reset register (RCC_APB2RSTR) 80
6.3.5 APB1 peripheral reset register (RCC_APB1RSTR) 82
6.3.6 AHB peripheral clock enable register (RCC_AHBENR) 84
6.3.7 APB2 peripheral clock enable register (RCC_APB2ENR) 85
6.3.8 APB1 peripheral clock enable register (RCC_APB1ENR) 87
6.3.9 Backup domain control register (RCC_BDCR) 89
6.3.10 Control/status register (RCC_CSR) 90
6.3.11 RCC register map 92

7 General-purpose and alternate-function I/Os (GPIOs and AFIOs) . . . 94
7.1 GPIO functional description 94
7.1.1 General-purpose I/O (GPIO) 96
7.1.2 Atomic bit set or reset 96
7.1.3 External interrupt/wakeup lines 97
7.1.4 Alternate functions (AF) 97
7.1.5 Software remapping of I/O alternate functions 97
7.1.6 GPIO locking mechanism 97
7.1.7 Input configuration 98
7.1.8 Output configuration 98
7.1.9 Alternate function configuration 99
7.1.10 Analog input configuration 100
7.2 GPIO registers 101
7.2.1 Port configuration register low (GPIOx_CRL) (x=A..G) 101
7.2.2 Port configuration register high (GPIOx_CRH) (x=A..G) 102
7.2.3 Port input data register (GPIOx_IDR) (x=A..G) 102
7.2.4 Port output data register (GPIOx_ODR) (x=A..G) 103
7.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G) 103
7.2.6 Port bit reset register (GPIOx_BRR) (x=A..G) 103
7.2.7 Port configuration lock register (GPIOx_LCKR) (x=A..G) 104

7.3 Alternate function I/O and debug configuration (AFIO) 105
7.3.1 Using OSC32_IN/OSC32_OUT pins as GPIO ports PC14/PC15 105
7.3.2 Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1 105
7.3.3 CAN alternate function remapping 105
7.3.4 JTAG/SWD alternate function remapping 105
7.3.5 ADC alternate function remapping 107
7.3.6 Timer alternate function remapping 107
7.3.7 USART Alternate function remapping 109
7.3.8 I2C 1 alternate function remapping 110
7.3.9 SPI 1 alternate function remapping 110
7.4 AFIO registers 110
7.4.1 Event control register (AFIO_EVCR) 110
7.4.2 AF remap and debug I/O configuration register (AFIO_MAPR) 111
7.4.3 External interrupt configuration register 1 (AFIO_EXTICR1) 114
7.4.4 External interrupt configuration register 2 (AFIO_EXTICR2) 114
7.4.5 External interrupt configuration register 3 (AFIO_EXTICR3) 115
7.4.6 External interrupt configuration register 4 (AFIO_EXTICR4) 115
7.5 GPIO and AFIO register maps 115

8 Interrupts and events 117
8.1 Nested vectored interrupt controller (NVIC) 117
8.1.1 SysTick calibration value register 117
8.1.2 Interrupt and exception vectors 117
8.2 External interrupt/event controller (EXTI) 120
8.2.1 Main features 120
8.2.2 Block diagram 120
8.2.3 Wakeup event management 121
8.2.4 Functional description 121
8.2.5 External interrupt/event line mapping 122
8.3 EXTI registers 124
8.3.1 Interrupt mask register (EXTI_IMR) 124
8.3.2 Event mask register (EXTI_EMR) 124
8.3.3 Rising trigger selection register (EXTI_RTSR) 125
8.3.4 Falling trigger selection register (EXTI_FTSR) 125
8.3.5 Software interrupt event register (EXTI_SWIER) 126
8.3.6 Pending register (EXTI_PR) 126

8.3.7 EXTI register map 127

9 DMA controller (DMA) 128

9.1 DMA introduction 128

9.2 DMA main features 128

9.3 DMA functional description 129

9.3.1 DMA transactions 129

9.3.2 Arbiter 130

9.3.3 DMA channels 130

9.3.4 Programmable data width, data alignment and endians 131

9.3.5 Error management 133

9.3.6 Interrupts 133

9.3.7 DMA request mapping 133

9.4 DMA registers 137

9.4.1 DMA interrupt status register (DMA_ISR) 137

9.4.2 DMA interrupt flag clear register (DMA_IFCR) 138

9.4.3 DMA channel x configuration register (DMA_CCRx) (x = 1 ..7) 139

9.4.4 DMA channel x number of data register (DMA_CNDTRx) (x = 1 ..7) . 140

9.4.5 DMA channel x peripheral address register (DMA_CPARx) (x = 1 ..7) 141

9.4.6 DMA channel x memory address register (DMA_CMARx) (x = 1 ..7) . 141

9.4.7 DMA register map 141

10 Analog-to-digital converter (ADC) 144

10.1 ADC introduction 144

10.2 ADC main features 144

10.3 ADC functional description 145

10.3.1 ADC on-off control 146

10.3.2 ADC clock 146

10.3.3 Channel selection 146

10.3.4 Single conversion mode 147

10.3.5 Continuous conversion mode 147

10.3.6 Timing diagram 147

10.3.7 Analog watchdog 148

10.3.8 Scan mode 149

10.3.9 Injected channel management 149

10.3.10 Discontinuous mode 150

10.4 Calibration 151
10.5 Data alignment 151
10.6 Channel-by-channel programmable sample time 152
10.7 Conversion on external trigger 152
10.8 DMA request 154
10.9 Dual ADC mode 155
10.9.1 Injected simultaneous mode 157
10.9.2 Regular simultaneous mode 157
10.9.3 Fast interleaved mode 158
10.9.4 Slow interleaved mode 158
10.9.5 Alternate trigger mode 159
10.9.6 Independent mode 160
10.9.7 Combined regular/injected simultaneous mode 160
10.9.8 Combined regular simultaneous + alternate trigger mode 160
10.9.9 Combined injected simultaneous + interleaved 161
10.10 Temperature sensor 162
10.11 ADC interrupts 163
10.12 ADC registers 164
10.12.1 ADC status register (ADC_SR) 164
10.12.2 ADC control register 1 (ADC_CR1) 165
10.12.3 ADC control register 2 (ADC_CR2) 167
10.12.4 ADC sample time register 1 (ADC_SMPR1) 170
10.12.5 ADC sample time register 2 (ADC_SMPR2) 171
10.12.6 ADC injected channel data offset register x (ADC_JOFRx)(x=1..4) . . 171
10.12.7 ADC watchdog high threshold register (ADC_HTR) 172
10.12.8 ADC watchdog low threshold register (ADC_LTR) 172
10.12.9 ADC regular sequence register 1 (ADC_SQR1) 172
10.12.10 ADC regular sequence register 2 (ADC_SQR2) 173
10.12.11 ADC regular sequence register 3 (ADC_SQR3) 174
10.12.12 ADC injected sequence register (ADC_JSQR) 174
10.12.13 ADC injected data register x (ADC_JDRx) (x= 1..4) 175
10.12.14 ADC regular data register (ADC_DR) 175
10.12.15 ADC register map 176

11 Digital-to-analog converter (DAC) 178
11.1 DAC introduction 178

11.2 DAC main features 178
11.3 DAC functional description 180
11.3.1 DAC channel enable 180
11.3.2 DAC output buffer enable 180
11.3.3 DAC data format 180
11.3.4 DAC conversion 181
11.3.5 DAC output voltage 182
11.3.6 DAC trigger selection 182
11.3.7 DMA request 183
11.3.8 Noise generation 183
11.3.9 Triangle-wave generation 184
11.4 Dual DAC channel conversion 185
11.4.1 Independent trigger without wave generation 185
11.4.2 Independent trigger with same LFSR generation 186
11.4.3 Independent trigger with different LFSR generation 186
11.4.4 Independent trigger with same triangle generation 186
11.4.5 Independent trigger with different triangle generation 187
11.4.6 Simultaneous software start 187
11.4.7 Simultaneous trigger without wave generation 187
11.4.8 Simultaneous trigger with same LFSR generation 188
11.4.9 Simultaneous trigger with different LFSR generation 188
11.4.10 Simultaneous trigger with same triangle generation 188
11.4.11 Simultaneous trigger with different triangle generation 189
11.5 DAC registers 189
11.5.1 DAC control register (DAC_CR) 189
11.5.2 DAC software trigger register (DAC_SWTRIGR) 192
11.5.3 DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1) 193
11.5.4 DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1) 193
11.5.5 DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1) 193
11.5.6 DAC channel2 12-bit right aligned data holding register (DAC_DHR12R2) 194
11.5.7 DAC channel2 12-bit left aligned data holding register (DAC_DHR12L2) 194
11.5.8 DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2) 195

11.5.9 Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD) . . . 195
11.5.10 DUAL DAC 12-bit left aligned data holding register (DAC_DHR12LD) . . . 196
11.5.11 DUAL DAC 8-bit right aligned data holding register (DAC_DHR8RD) . . . 196
11.5.12 DAC channel1 data output register (DAC_DOR1) . . . 197
11.5.13 DAC channel2 data output register (DAC_DOR2) . . . 197
11.5.14 DAC register map . . . 198

12 Advanced-control timers (TIM1&TIM8) . . . 199

12.1 TIM1&TIM8 introduction . . . 199
12.2 TIM1&TIM8 main features . . . 199
12.3 TIM1&TIM8 functional description . . . 202
12.3.1 Time-base unit . . . 202
12.3.2 Counter modes . . . 203
12.3.3 Repetition counter . . . 211
12.3.4 Clock selection . . . 213
12.3.5 Capture/compare channels . . . 215
12.3.6 Input capture mode . . . 217
12.3.7 PWM input mode . . . 218
12.3.8 Forced output mode . . . 219
12.3.9 Output compare mode . . . 220
12.3.10 PWM mode . . . 221
12.3.11 Complementary outputs and dead-time insertion . . . 224
12.3.12 Using the break function . . . 225
12.3.13 Clearing the OCxREF signal on an external event . . . 228
12.3.14 6-step PWM generation . . . 229
12.3.15 One-pulse mode . . . 230
12.3.16 Encoder interface mode . . . 231
12.3.17 Timer input XOR function . . . 234
12.3.18 Interfacing with Hall sensors . . . 234
12.3.19 TIMx and external trigger synchronization . . . 236
12.3.20 Timer synchronization . . . 239
12.3.21 Debug mode . . . 239
12.4 TIM1&TIM8 registers . . . 240
12.4.1 Control register 1 (TIMx_CR1) . . . 240
12.4.2 Control register 2 (TIMx_CR2) . . . 241

12.4.3 Slave mode control register (TIMx_SMCR) 243
12.4.4 DMA/Interrupt enable register (TIMx_DIER) 245
12.4.5 Status register (TIMx_SR) 246
12.4.6 Event generation register (TIMx_EGR) 249
12.4.7 Capture/compare mode register 1 (TIMx_CCMR1) 250
12.4.8 Capture/compare mode register 2 (TIMx_CCMR2) 253
12.4.9 Capture/compare enable register (TIMx_CCER) 254
12.4.10 Counter (TIMx_CNT) 257
12.4.11 Prescaler (TIMx_PSC) 257
12.4.12 Auto-reload register (TIMx_ARR) 257
12.4.13 Repetition counter register (TIMx_RCR) 258
12.4.14 Capture/compare register 1 (TIMx_CCR1) 258
12.4.15 Capture/compare register 2 (TIMx_CCR2) 259
12.4.16 Capture/compare register 3 (TIMx_CCR3) 259
12.4.17 Capture/compare register 4 (TIMx_CCR4) 260
12.4.18 Break and dead-time register (TIMx_BDTR) 260
12.4.19 DMA control register (TIMx_DCR) 262
12.4.20 DMA address for full transfer (TIMx_DMAR) 264
12.4.21 TIM1&TIM8 register map 264

13 General-purpose timer (TIMx) 266

13.1 TIMx introduction 266
13.2 TIMx main features 267
13.3 TIMx functional description 268
13.3.1 Time-base unit 268
13.3.2 Counter modes 270
13.3.3 Clock selection 278
13.3.4 Capture/compare channels 281
13.3.5 Input capture mode 283
13.3.6 PWM input mode 284
13.3.7 Forced output mode 284
13.3.8 Output compare mode 285
13.3.9 PWM mode 286
13.3.10 One pulse mode 289
13.3.11 Clearing the OCxREF signal on an external event 290
13.3.12 Encoder interface mode 291
13.3.13 Timer input XOR function 293

13.3.14 Timers and external trigger synchronization 293
13.3.15 Timer synchronization 296
13.3.16 Debug mode 301
13.4 TIMx registers 302
13.4.1 Control register 1 (TIMx_CR1) 302
13.4.2 Control register 2 (TIMx_CR2) 303
13.4.3 Slave mode control register (TIMx_SMCR) 304
13.4.4 DMA/Interrupt enable register (TIMx_DIER) 307
13.4.5 Status register (TIMx_SR) 308
13.4.6 Event generation register (TIMx_EGR) 309
13.4.7 Capture/compare mode register 1 (TIMx_CCMR1) 310
13.4.8 Capture/compare mode register 2 (TIMx_CCMR2) 314
13.4.9 Capture/compare enable register (TIMx_CCER) 315
13.4.10 Counter (TIMx_CNT) 316
13.4.11 Prescaler (TIMx_PSC) 316
13.4.12 Auto-reload register (TIMx_ARR) 317
13.4.13 Capture/compare register 1 (TIMx_CCR1) 317
13.4.14 Capture/compare register 2 (TIMx_CCR2) 317
13.4.15 Capture/compare register 3 (TIMx_CCR3) 318
13.4.16 Capture/compare register 4 (TIMx_CCR4) 318
13.4.17 DMA control register (TIMx_DCR) 319
13.4.18 DMA address for full transfer (TIMx_DMAR) 320
13.4.19 TIMx register map 320

14 Basic timer (TIM6&7) 322
14.1 TIM6&7 introduction 322
14.2 TIM6&TIM7 main features 322
14.3 TIM6&TIM7 functional description 323
14.3.1 Time-base unit 323
14.3.2 Counting mode 325
14.3.3 Clock source 327
14.3.4 Debug mode 328
14.4 TIM6&TIM7 registers 328
14.4.1 Control register 1 (TIMx_CR1) 328
14.4.2 Control register 2 (TIMx_CR2) 330
14.4.3 DMA/Interrupt enable register (TIMx_DIER) 330
14.4.4 Status register (TIMx_SR) 331

14.4.5 Event generation register (TIMx_EGR) 331
14.4.6 Counter (TIMx_CNT) 332
14.4.7 Prescaler (TIMx_PSC) 332
14.4.8 Auto-reload register (TIMx_ARR) 332
14.4.9 TIM6&7 register map 333

15 Real-time clock (RTC) 334
15.1 RTC introduction 334
15.2 RTC main features 335
15.3 RTC functional description 335
15.3.1 Overview 335
15.3.2 Resetting RTC registers 337
15.3.3 Reading RTC registers 337
15.3.4 Configuring RTC registers 337
15.3.5 RTC flag assertion 338
15.4 RTC registers 339
15.4.1 RTC control register high (RTC_CRH) 339
15.4.2 RTC control register low (RTC_CRL) 340
15.4.3 RTC prescaler load register (RTC_PRLH / RTC_PRLL) 341
15.4.4 RTC prescaler divider register (RTC_DIVH / RTC_DIVL) 342
15.4.5 RTC counter register (RTC_CNTH / RTC_CNTL) 343
15.4.6 RTC alarm register high (RTC_ALRH / RTC_ALRL) 344
15.4.7 RTC register map 345

16 Independent watchdog (IWDG) 346
16.1 IWDG introduction 346
16.2 IWDG main features 346
16.3 IWDG functional description 346
16.3.1 Hardware watchdog 347
16.3.2 Register access protection 347
16.3.3 Debug mode 347
16.4 IWDG registers 348
16.4.1 Key register (IWDG_KR) 348
16.4.2 Prescaler register (IWDG_PR) 348
16.4.3 Reload register (IWDG_RLR) 349
16.4.4 Status register (IWDG_SR) 350

16.4.5 IWDG register map 351

17 Window watchdog (WWDG) 352

17.1 WWDG introduction 352
17.2 WWDG main features 352
17.3 WWDG functional description 352
17.4 How to program the watchdog timeout 354
17.5 Debug mode 355
17.6 Debug registers 355
17.6.1 Control Register (WWDG_CR) 355
17.6.2 Configuration register (WWDG_CFR) 355
17.6.3 Status register (WWDG_SR) 356
17.6.4 WWDG register map 356

18 Flexible static memory controller (FSMC) 357

18.1 FSMC main features 357
18.2 Block diagram 358
18.3 AHB interface 359
18.3.1 Supported memories and transactions 359
18.4 External device address mapping 360
18.4.1 NOR/PSRAM address mapping 360
18.4.2 NAND/PC Card address mapping 361
18.5 NOR Flash/PSRAM controller 362
18.5.1 External memory interface signals 363
18.5.2 Supported memories and transactions 365
18.5.3 General timing rules 366
18.5.4 NOR Flash/PSRAM controller timing diagrams 366
18.5.5 Synchronous burst read 379
18.5.6 NOR/PSRAM controller registers 385
18.6 NAND Flash/PC Card controller 390
18.6.1 External memory interface signals 391
18.6.2 NAND Flash / PC Card supported memories and transactions 392
18.6.3 Timing diagrams for NAND, ATA and PC Card 393
18.6.4 NAND Flash operations 394
18.6.5 NAND Flash pre-wait functionality 395
18.6.6 Error correction code computation ECC (NAND Flash) 396

18.6.7 NAND Flash/PC Card controller registers 396

19 SDIO interface (SDIO) 403

19.1 SDIO main features 403
19.2 SDIO bus topology 404
19.3 SDIO functional description 406
19.3.1 SDIO adapter 407
19.3.2 SDIO AHB Interface 417
19.4 Card functional description 418
19.4.1 Card identification mode 418
19.4.2 Card reset 418
19.4.3 Operating voltage range validation 418
19.4.4 Card identification process 419
19.4.5 Block write 420
19.4.6 Block read 420
19.4.7 Stream access, stream write and stream read (MultiMediaCard only) 421
19.4.8 Erase: group erase and sector erase 422
19.4.9 Wide bus selection or deselection 422
19.4.10 Protection management 423
19.4.11 Card status register 426
19.4.12 SD status register 429
19.4.13 SD I/O mode 433
19.4.14 Commands and responses 434
19.5 Response formats 437
19.5.1 R1 (normal response command) 437
19.5.2 R1b 438
19.5.3 R2 (CID, CSD register) 438
19.5.4 R3 (OCR register) 438
19.5.5 R4 (Fast I/O) 439
19.5.6 R4b 439
19.5.7 R5 (interrupt request) 440
19.5.8 R6 440
19.6 SDIO I/O card-specific operations 441
19.6.1 SDIO I/O read wait operation by SDIO_D2 signalling 441
19.6.2 SDIO read wait operation by stopping SDIO_CK 441
19.6.3 SDIO suspend/resume operation 442

19.6.4 SDIO interrupts 442
19.7 CE-ATA specific operations 442
19.7.1 Command completion signal disable 442
19.7.2 Command completion signal enable 442
19.7.3 CE-ATA interrupt 443
19.7.4 Aborting CMD61 443
19.8 HW flow control 443
19.9 SDIO registers 443
19.9.1 SDIO power control register (SDIO_POWER) 443
19.9.2 SDI clock control register (SDIO_CLKCR) 444
19.9.3 SDIO argument register (SDIO_ARG) 445
19.9.4 SDIO command register (SDIO_CMD) 445
19.9.5 SDIO command response register (SDIO_RESPCMD) 446
19.9.6 SDIO response 0..4 register (SDIO_RESPx) 446
19.9.7 SDIO data timer register (SDIO_DTIMER) 447
19.9.8 SDIO data length register (SDIO_DLEN) 447
19.9.9 SDIO data control register (SDIO_DCTRL) 448
19.9.10 SDIO data counter register (SDIO_DCOUNT) 449
19.9.11 SDIO status register (SDIO_STA) 450
19.9.12 SDIO interrupt clear register (SDIO_ICR) 451
19.9.13 SDIO mask register (SDIO_MASK) 453
19.9.14 SDIO FIFO counter register (SDIO_FIFOCNT) 455
19.9.15 SDIO data FIFO register (SDIO_FIFO) 456
19.9.16 SDIO register map 456

20 USB full speed device interface (USB) 458
20.1 USB introduction 458
20.2 USB main features 458
20.3 USB functional description 458
20.3.1 Description of USB blocks 460
20.4 Programming considerations 461
20.4.1 Generic USB device programming 461
20.4.2 System and power-on reset 462
20.4.3 Double-buffered endpoints 468
20.4.4 Isochronous transfers 470
20.4.5 Suspend/Resume events 471

20.5 USB registers 473
20.5.1 Common registers 473
20.5.2 Endpoint-specific registers 480
20.5.3 Buffer descriptor table 483
20.5.4 USB register map 487

21 Controller area network (bxCAN) 489
21.1 bxCAN introduction 489
21.2 bxCAN main features 489
21.2.1 General description 490
21.3 bxCAN operating modes 492
21.3.1 Initialization mode 492
21.3.2 Normal mode 493
21.3.3 Sleep mode (low power) 493
21.3.4 Test mode 493
21.3.5 Silent mode 494
21.3.6 Loop back mode 494
21.3.7 Loop back combined with silent mode 494
21.4 bxCAN functional description 495
21.4.1 Transmission handling 495
21.4.2 Time triggered communication mode 496
21.4.3 Reception handling 497
21.4.4 Identifier filtering 498
21.4.5 Message storage 502
21.4.6 Error management 504
21.4.7 Bit timing 504
21.5 bxCAN interrupts 506
21.6 CAN registers 508
21.6.1 Register access protection 508
21.6.2 CAN control and status registers 508
21.6.3 Mailbox registers 519
21.6.4 CAN filter registers 524
21.6.5 bxCAN register map 527

22 Serial peripheral interface (SPI) 531
22.1 SPI introduction 531

22.2 SPI and I^{2}S main features 532
22.2.1 SPI features 532
22.2.2 I^{2}S features 533
22.3 SPI functional description 534
22.3.1 General description 534
22.3.2 SPI slave mode 538
22.3.3 SPI master mode 538
22.3.4 Simplex communication 539
22.3.5 Status flags 540
22.3.6 CRC calculation 540
22.3.7 SPI communication using DMA (direct memory addressing) 541
22.3.8 Error flags 542
22.3.9 Disabling the SPI 543
22.3.10 SPI interrupts 543
22.4 I^{2}S functional description 543
22.4.1 General description 543
22.4.2 Supported audio protocols 545
22.4.3 Clock generator 552
22.4.4 I^{2}S master mode 553
22.4.5 I^{2}S slave mode 554
22.4.6 Status flags 556
22.4.7 Error flags 556
22.4.8 I^{2}S interrupts 557
22.4.9 DMA features 557
22.5 SPI and I^{2}S registers 557
22.5.1 SPI Control Register 1 (SPI_CR1) (not used in I^{2}S mode) 557
22.5.2 SPI control register 2 (SPI_CR2) 560
22.5.3 SPI status register (SPI_SR) 561
22.5.4 SPI data register (SPI_DR) 562
22.5.5 SPI CRC polynomial register (SPI_CRCPR) (not used in I^{2}S mode) 563
22.5.6 SPI Rx CRC register (SPI_RXCRCR) (not used in I^{2}S mode) 563
22.5.7 SPI Tx CRC register (SPI_TXCRCR) (not used in I^{2}S mode) 564
22.5.8 SPI_I^{2}S configuration register (SPI_I2SCFGR) 564
22.5.9 SPI_I^{2}S Prescaler register (SPI_I2SPR) 566
22.5.10 SPI register map 567

23 Inter-integrated circuit (I^2C) interface 568

23.1 I^2C Introduction 568

23.2 I^2C main features 568

23.3 I^2C functional description 569

23.3.1 Mode selection 569

23.3.2 I2C slave mode 571

23.3.3 I2C master mode 574

23.3.4 Error conditions 577

23.3.5 SDA/SCL line control 578

23.3.6 SMBus 578

23.3.7 DMA requests 581

23.3.8 Packet error checking 582

23.4 I^2C interrupts 583

23.5 I^2C debug mode 584

23.6 I^2C registers 584

23.6.1 Control register 1(I2C_CR1) 584

23.6.2 Control register 2 (I2C_CR2) 587

23.6.3 Own address register 1 (I2C_OAR1) 588

23.6.4 Own address register 2 (I2C_OAR2) 589

23.6.5 Data register (I2C_DR) 589

23.6.6 Status register 1 (I2C_SR1) 590

23.6.7 Status register 2 (I2C_SR2) 593

23.6.8 Clock control register (I2C_CCR) 594

23.6.9 TRISE Register (I2C_TRISE) 595

23.6.10 I2C register map 596

24 Universal synchronous asynchronous receiver transmitter (USART) 597

24.1 USART introduction 597

24.2 USART main features 597

24.3 USART functional description 598

24.3.1 USART character description 601

24.3.2 Transmitter 602

24.3.3 Receiver 604

24.3.4 Fractional baud rate generation 608

24.3.5 Multiprocessor communication 610

24.3.6 Parity control 611
24.3.7 LIN (local interconnection network) mode 612
24.3.8 USART synchronous mode 614
24.3.9 Single wire half duplex communication 616
24.3.10 Smartcard 617
24.3.11 IrDA SIR ENDEC block 619
24.3.12 Continuous communication using DMA 620
24.3.13 Hardware flow control 622
24.4 USART interrupts 624
24.5 USART mode configuration 625
24.6 USART registers 625
24.6.1 Status register (USART_SR) 625
24.6.2 Data register (USART_DR) 628
24.6.3 Baud rate register (USART_BRR) 628
24.6.4 Control register 1 (USART_CR1) 629
24.6.5 Control register 2 (USART_CR2) 631
24.6.6 Control register 3 (USART_CR3) 633
24.6.7 Guard time and prescaler register (USART_GTPR) 634
24.6.8 USART register map 636

25 Device electronic signature 637
25.1 Memory size registers 637
25.1.1 Flash size register 637
25.2 Unique device ID register (96 bits) 638

26 Debug support (DBG) 640
26.1 Overview 640
26.2 Reference ARM documentation 642
26.3 SWJ debug port (serial wire and JTAG) 642
26.3.1 Mechanism to select the JTAG-DP or the SW-DP 642
26.4 Pinout and debug port pins 643
26.4.1 SWJ debug port pins 643
26.4.2 Flexible SWJ-DP pin assignment 643
26.4.3 Internal pull-up and pull-down on JTAG pins 644
26.4.4 Using serial wire and releasing the unused debug pins as GPIOs . . . 645
26.5 STM32F10xxx JTAG TAP connection 645

26.6 ID codes and locking mechanism 646
26.6.1 MCU device ID code 646
26.6.2 Boundary scan TAP 647
26.6.3 Cortex-M3 TAP 647
26.6.4 Cortex-M3 JEDEC-106 ID code 648
26.7 JTAG debug port 648
26.8 SW debug port 650
26.8.1 SW protocol introduction 650
26.8.2 SW protocol sequence 650
26.8.3 SW-DP state machine (Reset, idle states, ID code) 651
26.8.4 DP and AP read/write accesses 651
26.8.5 SW-DP registers 652
26.8.6 SW-AP registers 652
26.9 AHB-AP (AHB Access Port) - valid for both JTAG-DP or SW-DP 653
26.10 Core debug 653
26.11 Capability of the debugger host to connect under system reset 654
26.12 FPB (Flash patch breakpoint) 655
26.13 DWT (data watchpoint trigger) 655
26.14 ITM (instrumentation trace macrocell) 655
26.14.1 General description 655
26.14.2 Timestamp packets, synchronization and overflow packets 656
26.15 MCU debug component (MCUDBG) 657
26.15.1 Debug support for low-power modes 657
26.15.2 Debug support for timers, watchdog, bxCAN and I^2C 658
26.15.3 Debug MCU configuration register 658
26.16 TPIU (trace port interface unit) 660
26.16.1 Introduction 660
26.16.2 TRACE pin assignment 661
26.16.3 TPUI formatter 663
26.16.4 TPUI frame synchronization packets 664
26.16.5 Emission of synchronization frame packet 664
26.16.6 Synchronous mode 664
26.16.7 Asynchronous mode 665
26.16.8 TRACECLKIN connection inside STM32F10xxx 665
26.16.9 TPIU registers 665
26.16.10 Example of configuration 666

26.17 DBG register map 667

27 Revision history 668

List of tables

Table 1. Register boundary addresses . . . 35
Table 2. Flash module organization (low-density devices) . . . 38
Table 3. Flash module organization (medium-density devices) . . . 38
Table 4. Flash module organization (high-density devices) . . . 39
Table 5. Boot modes . . . 41
Table 6. CRC calculation unit register map and reset values . . . 44
Table 7. Low-power mode summary . . . 49
Table 8. Sleep-now . . . 51
Table 9. Sleep-on-exit . . . 51
Table 10. Stop mode . . . 52
Table 11. Standby mode . . . 53
Table 12. PWR - register map and reset values . . . 57
Table 13. BKP register map and reset values . . . 63
Table 14. RCC - register map and reset values . . . 92
Table 15. Port bit configuration table . . . 96
Table 16. Output MODE bits . . . 96
Table 17. CAN1 alternate function remapping . . . 105
Table 18. Debug interface signals . . . 106
Table 19. Debug port mapping . . . 106
Table 20. ADC1 external trigger injected conversion alternate function remapping . . . 107
Table 21. ADC1 external trigger regular conversion alternate function remapping . . . 107
Table 22. ADC2 external trigger injected conversion alternate function remapping . . . 107
Table 23. ADC2 external trigger regular conversion alternate function remapping . . . 107
Table 24. Timer 5 alternate function remapping . . . 107
Table 25. Timer 4 alternate function remapping . . . 108
Table 26. Timer 3 alternate function remapping . . . 108
Table 27. Timer 2 alternate function remapping . . . 108
Table 28. Timer 1 alternate function remapping . . . 109
Table 29. USART3 remapping . . . 109
Table 30. USART2 remapping . . . 109
Table 31. USART1 remapping . . . 110
Table 32. I2C1 remapping . . . 110
Table 33. SPI1 remapping . . . 110
Table 34. GPIO register map and reset values . . . 116
Table 35. AFIO register map and reset values . . . 116
Table 36. Vector table for other STM32F10xxx devices . . . 117
Table 37. External interrupt/event controller register map and reset values . . . 127
Table 39. DMA interrupt requests . . . 133
Table 40. Summary of DMA1 requests for each channel . . . 135
Table 41. Summary of DMA2 requests for each channel . . . 136
Table 42. DMA - register map and reset values . . . 141
Table 43. ADC pins . . . 146
Table 44. Analog watchdog channel selection . . . 148
Table 45. External trigger for regular channels for ADC1 and ADC2 . . . 153
Table 46. External trigger for injected channels for ADC1 and ADC2 . . . 153
Table 47. External trigger for regular channels for ADC3 . . . 154
Table 48. External trigger for injected channels for ADC3 . . . 154
Table 49. ADC interrupts . . . 163

Table 50. ADC register map and reset values . . . 176
Table 51. DAC pins. . . . 179
Table 52. External triggers . . . 182
Table 53. DAC register map . . . 198
Table 54. Counting direction versus encoder signals . . . 232
Table 55. TIMx Internal trigger connection . . . 245
Table 56. Output control bits for complementary OCx and OCxN channels with break feature . . . 256
Table 57. TIM1&TIM8 Register map and reset values . . . 264
Table 58. Counting direction versus encoder signals . . . 292
Table 59. TIMx Internal trigger connection . . . 306
Table 60. Output control bit for standard OCx channels. . . . 316
Table 61. TIMx register map and reset values . . . 320
Table 62. TIM6&7 - register map and reset values. . . . 333
Table 63. RTC - register map and reset values . . . 345
Table 64. Watchdog timeout period (with 40 kHz input clock) . . . 347
Table 65. IWDG register map and reset values . . . 351
Table 66. WWDG register map and reset values . . . 356
Table 67. NOR/PSRAM bank selection . . . 360
Table 68. External memory address. . . . 361
Table 69. Memory mapping and timing registers . . . 361
Table 70. NAND bank selections . . . 362
Table 71. Programmable NOR/PSRAM access parameters . . . 363
Table 72. Nonmuxed I/O NOR Flash . . . 363
Table 73. Muxed I/O NOR Flash . . . 364
Table 74. PSRAM . . . 364
Table 75. NOR Flash/PSRAM supported memories and transactions . . . 365
Table 76. FSMC_BCRx bit fields . . . 367
Table 77. FSMC_TCRx bit fields . . . 368
Table 78. FSMC_BCRx bit fields . . . 369
Table 79. FSMC_TCRx bit fields . . . 370
Table 80. FSMC_BWTRx bit fields . . . 370
Table 81. FSMC_BCRx bit fields . . . 372
Table 82. FSMC_TCRx bit fields . . . 373
Table 83. FSMC_BWTRx bit fields . . . 373
Table 84. FSMC_BCRx bit fields . . . 375
Table 85. FSMC_TCRx bit fields . . . 375
Table 86. FSMC_BWTRx bit fields . . . 375
Table 87. FSMC_BCRx bit fields . . . 377
Table 88. FSMC_TCRx bit fields . . . 377
Table 89. FSMC_BWTRx bit fields . . . 377
Table 90. FSMC_BCRx bit fields . . . 379
Table 91. FSMC_TCRx bit fields . . . 379
Table 92. FSMC_BCRx bit fields . . . 381
Table 93. FSMC_TCRx bit fields . . . 382
Table 94. FSMC_BCRx bit fields . . . 384
Table 95. FSMC_TCRx bit fields . . . 384
Table 96. Programmable NAND/PC Card access parameters . . . 391
Table 97. 8-bit NAND Flash . . . 391
Table 98. 16-bit NAND Flash . . . 392
Table 99. 16-bit PC Card . . . 392
Table 100. Supported memories and transactions . . . 393
Table 101. ECC result relevant bits . . . 402

Table 102. SDIO I/O definitions . . . 407
Table 103. Command format . . . 411
Table 104. Short response format . . . 412
Table 105. Long response format . . . 412
Table 106. Command path status flags . . . 412
Table 107. Data token format . . . 415
Table 108. Transmit FIFO status flags . . . 416
Table 109. Receive FIFO status flags . . . 417
Table 110. Card status . . . 427
Table 111. SD status . . . 429
Table 112. Speed class code field . . . 431
Table 113. Performance move field . . . 431
Table 114. AU_SIZE field . . . 431
Table 115. Maximum AU size . . . 432
Table 116. Erase size field . . . 432
Table 117. Erase timeout field . . . 432
Table 118. Erase offset field . . . 433
Table 119. Block-oriented write commands . . . 435
Table 120. Block-oriented write protection commands . . . 436
Table 121. Erase commands . . . 436
Table 122. I/O mode commands . . . 436
Table 123. Lock card . . . 437
Table 124. Application-specific commands . . . 437
Table 125. R1 response . . . 438
Table 126. R2 response . . . 438
Table 127. R3 response . . . 439
Table 128. R4 response . . . 439
Table 129. R4b response . . . 439
Table 130. R5 response . . . 440
Table 131. R6 response . . . 440
Table 132. Response type and SDIO_RESPx registers . . . 447
Table 133. SDIO register map . . . 456
Table 134. Double-buffering buffer flag definition . . . 469
Table 135. Bulk double-buffering memory buffers usage . . . 469
Table 136. Isochronous memory buffers usage . . . 471
Table 137. Resume event detection . . . 472
Table 138. Reception status encoding . . . 483
Table 139. Endpoint type encoding . . . 483
Table 140. Endpoint kind meaning . . . 483
Table 141. Transmission status encoding . . . 483
Table 142. Definition of allocated buffer memory . . . 486
Table 143. USB register map and reset values . . . 487
Table 144. Transmit mailbox mapping . . . 503
Table 145. Receive mailbox mapping . . . 503
Table 146. bxCAN register map and reset values . . . 528
Table 147. SPI interrupt requests . . . 543
Table 148. I^2S interrupt requests . . . 557
Table 149. SPI register map and reset values . . . 567
Table 150. SMBus vs. I2C . . . 579
Table 151. I2C Interrupt requests . . . 583
Table 152. I2C register map and reset values . . . 596
Table 153. Noise detection from sampled data . . . 607

Table 154. Error calculation for programmed baud rates 609
Table 155. Frame formats 611
Table 156. USART interrupt requests. 624
Table 157. USART modes configuration 625
Table 158. USART register map and reset values 636
Table 159. SWJ debug port pins 643
Table 160. Flexible SWJ-DP pin assignment 644
Table 161. JTAG debug port data registers 648
Table 162. 32-bit debug port registers addressed through the shifted value A[3:2] 649
Table 163. Packet request (8-bits) 650
Table 164. ACK response (3 bits). 651
Table 165. DATA transfer (33 bits) 651
Table 166. SW-DP registers 652
Table 167. Cortex-M3 AHB-AP registers 653
Table 168. Core debug registers 654
Table 169. Main ITM registers 656
Table 170. Asynchronous TRACE pin assignment. 661
Table 171. Synchronous TRACE pin assignment 661
Table 172. Flexible TRACE pin assignment 662
Table 173. Important TPIU registers. 665
Table 174. DBG register map and reset values 667
Table 175. Document revision history 668

List of figures

Figure 1. System architecture . . . 33
Figure 2. CRC calculation unit block diagram . . . 42
Figure 3. Power supply overview . . . 45
Figure 4. Power on reset/power down reset waveform . . . 48
Figure 5. PVD thresholds . . . 48
Figure 6. Reset circuit . . . 67
Figure 7. Clock tree . . . 68
Figure 8. HSE/ LSE clock sources . . . 69
Figure 9. Basic structure of a standard I/O port bit . . . 95
Figure 10. Basic structure of a five-volt tolerant I/O port bit . . . 95
Figure 11. Input floating/pull up/pull down configurations . . . 98
Figure 12. Output configuration . . . 99
Figure 13. Alternate function configuration . . . 100
Figure 14. High impedance-analog input configuration . . . 100
Figure 15. External interrupt/event controller block diagram . . . 121
Figure 16. External interrupt/event GPIO mapping . . . 123
Figure 17. DMA block diagram . . . 129
Figure 18. DMA1 request mapping . . . 134
Figure 19. DMA2 request mapping . . . 136
Figure 20. Single ADC block diagram . . . 145
Figure 21. Timing diagram . . . 148
Figure 22. Analog watchdog guarded area . . . 148
Figure 23. Injected conversion latency . . . 150
Figure 24. Calibration timing diagram . . . 151
Figure 25. Right alignment of data . . . 152
Figure 26. Left alignment of data . . . 152
Figure 27. Dual ADC block diagram . . . 156
Figure 28. Injected simultaneous mode on 4 channels . . . 157
Figure 29. Regular simultaneous mode on 16 channels . . . 158
Figure 30. Fast interleaved mode on 1 channel in continuous conversion mode . . . 158
Figure 31. Slow interleaved mode on 1 channel . . . 159
Figure 32. Alternate trigger: injected channel group of each ADC . . . 159
Figure 33. Alternate trigger: 4 injected channels (each ADC) in discontinuous model . . . 160
Figure 34. Alternate + Regular simultaneous . . . 161
Figure 35. Case of trigger occurring during injected conversion . . . 161
Figure 36. Interleaved single channel with injected sequence CH11, CH12 . . . 161
Figure 37. Temperature sensor and VREFINT channel block diagram . . . 162
Figure 38. DAC channel block diagram . . . 179
Figure 39. Data registers in single DAC channel mode . . . 181
Figure 40. Data registers in dual DAC channel mode . . . 181
Figure 41. Timing diagram for conversion with trigger disabled TEN = 0 . . . 182
Figure 42. DAC LFSR register calculation algorithm . . . 183
Figure 43. DAC conversion (SW trigger enabled) with LFSR wave generation . . . 184
Figure 44. DAC triangle wave generation . . . 184
Figure 45. DAC conversion (SW trigger enabled) with triangle wave generation . . . 185
Figure 46. Advanced-control timer block diagram . . . 201
Figure 47. Counter timing diagram with prescaler division change from 1 to 2 . . . 203
Figure 48. Counter timing diagram with prescaler division change from 1 to 4 . . . 203

Figure 49. Counter timing diagram, internal clock divided by 1 204
Figure 50. Counter timing diagram, internal clock divided by 2 204
Figure 51. Counter timing diagram, internal clock divided by 4 205
Figure 52. Counter timing diagram, internal clock divided by N. 205
Figure 53. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) 205
Figure 54. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) 206
Figure 55. Counter timing diagram, internal clock divided by 1 207
Figure 56. Counter timing diagram, internal clock divided by 2 207
Figure 57. Counter timing diagram, internal clock divided by 4 207
Figure 58. Counter timing diagram, internal clock divided by N. 208
Figure 59. Counter timing diagram, update event when repetition counter is not used. 208
Figure 60. Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6 209
Figure 61. Counter timing diagram, internal clock divided by 2 209
Figure 62. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 210
Figure 63. Counter timing diagram, internal clock divided by N. 210
Figure 64. Counter timing diagram, update event with ARPE=1 (counter underflow) 210
Figure 65. Counter timing diagram, Update event with ARPE=1 (counter overflow). 211
Figure 66. Update rate examples depending on mode and TIMx_RCR register settings 212
Figure 67. Control circuit in normal mode, internal clock divided by 1 213
Figure 68. TI2 external clock connection example. 213
Figure 69. Control circuit in external clock mode 1 214
Figure 70. External trigger input block 214
Figure 71. Control circuit in external clock mode 2 215
Figure 72. Capture/compare channel (example: channel 1 input stage) 216
Figure 73. Capture/compare channel 1 main circuit 216
Figure 74. Output stage of capture/compare channel (channel 1 to 3) 217
Figure 75. Output stage of capture/compare channel (channel 4). 217
Figure 76. PWM input mode timing 219
Figure 77. Output compare mode, toggle on OC1. 221
Figure 78. Edge-aligned PWM waveforms (ARR=8) 222
Figure 79. Center-aligned PWM waveforms (ARR=8) 223
Figure 80. Complementary output with dead-time insertion. 224
Figure 81. Dead-time waveforms with delay greater than the negative pulse. 224
Figure 82. Dead-time waveforms with delay greater than the positive pulse. 225
Figure 83. Output behavior in response to a break.. 227
Figure 84. Clearing TIMx OCxREF 228
Figure 85. 6-step generation, COM example (OSSR=1) 229
Figure 86. Example of one pulse mode. 230
Figure 87. Example of counter operation in encoder interface mode. 233
Figure 88. Example of encoder interface mode with TI1FP1 polarity inverted. 233
Figure 89. Example of hall sensor interface. 235
Figure 90. Control circuit in reset mode 236
Figure 91. Control circuit in gated mode 237
Figure 92. Control circuit in trigger mode. 238
Figure 93. Control circuit in external clock mode 2 + trigger mode 239
Figure 94. General-purpose timer block diagram 268
Figure 95. Counter timing diagram with prescaler division change from 1 to 2 269
Figure 96. Counter timing diagram with prescaler division change from 1 to 4 270
Figure 97. Counter timing diagram, internal clock divided by 1 271
Figure 98. Counter timing diagram, internal clock divided by 2 271
Figure 99. Counter timing diagram, internal clock divided by 4 271
Figure 100. Counter timing diagram, internal clock divided by N. 272

Figure 101. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded). 272
Figure 102. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded). 273
Figure 103. Counter timing diagram, internal clock divided by 1 274
Figure 104. Counter timing diagram, internal clock divided by 2 274
Figure 105. Counter timing diagram, internal clock divided by 4 274
Figure 106. Counter timing diagram, internal clock divided by N. 275
Figure 107. Counter timing diagram, Update event when repetition counter is not used 275
Figure 108. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6 276
Figure 109. Counter timing diagram, internal clock divided by 2 276
Figure 110. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 277
Figure 111. Counter timing diagram, internal clock divided by N. 277
Figure 112. Counter timing diagram, Update event with ARPE=1 (counter underflow). 277
Figure 113. Counter timing diagram, Update event with ARPE=1 (counter overflow) 278
Figure 114. Control circuit in normal mode, internal clock divided by 1 279
Figure 115. TI2 external clock connection example. 279
Figure 116. Control circuit in external clock mode 1 280
Figure 117. External trigger input block 280
Figure 118. Control circuit in external clock mode 2 281
Figure 119. Capture/compare channel (example: channel 1 input stage) 281
Figure 120. Capture/compare channel 1 main circuit 282
Figure 121. Output stage of capture/compare channel (channel 1). 282
Figure 122. PWM input mode timing 284
Figure 123. Output compare mode, toggle on OC1. 286
Figure 124. Edge-aligned PWM waveforms (ARR=8) 287
Figure 125. Center-aligned PWM waveforms (ARR=8) 288
Figure 126. Example of one pulse mode. 289
Figure 127. Clearing TIMx OCxREF 291
Figure 128. Example of counter operation in encoder interface mode. 292
Figure 129. Example of encoder interface mode with IC1FP1 polarity inverted. 293
Figure 130. Control circuit in reset mode 294
Figure 131. Control circuit in gated mode 295
Figure 132. Control circuit in trigger mode 295
Figure 133. Control circuit in external clock mode 2 + trigger mode 296
Figure 134. Master/Slave timer example 297
Figure 135. Gating timer 2 with OC1REF of timer 1 298
Figure 136. Gating timer 2 with Enable of timer 1 299
Figure 137. Triggering timer 2 with Update of timer 1 299
Figure 138. Triggering timer 2 with Enable of timer 1 300
Figure 139. Triggering timer 1 and 2 with timer 1 TI1 input. 301
Figure 140. Basic timer block diagram. 323
Figure 141. Counter timing diagram with prescaler division change from 1 to 2 324
Figure 142. Counter timing diagram with prescaler division change from 1 to 4 324
Figure 143. Counter timing diagram, internal clock divided by 1 325
Figure 144. Counter timing diagram, internal clock divided by 2 326
Figure 145. Counter timing diagram, internal clock divided by 4 326
Figure 146. Counter timing diagram, internal clock divided by N. 326
Figure 147. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded). 327
Figure 148. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded). 327
Figure 149. Control circuit in normal mode, internal clock divided by 1 328
Figure 150. RTC simplified block diagram 336

Figure 151. RTC second and alarm waveform example with PR=0003, ALARM=00004 338
Figure 152. RTC Overflow waveform example with PR=0003. 338
Figure 153. Independent watchdog block diagram 347
Figure 154. Watchdog block diagram 353
Figure 155. Window watchdog timing diagram 354
Figure 156. FSMC block diagram 358
Figure 157. FSMC memory banks 360
Figure 158. Mode1 read accesses. 366
Figure 159. Mode1 write accesses 367
Figure 160. ModeA read accesses 368
Figure 161. ModeA write accesses 369
Figure 162. Mode2/B read accesses 371
Figure 163. Mode2 write accesses 371
Figure 164. ModeB write accesses 372
Figure 165. ModeC read accesses 374
Figure 166. ModeC write accesses 374
Figure 167. ModeD read accesses 376
Figure 168. ModeD write accesses 376
Figure 169. Muxed read accesses. 378
Figure 170. Muxed write accesses 378
Figure 171. Synchronous multiplexed read mode - NOR, PSRAM (CRAM) 381
Figure 172. Synchronous multiplexed write mode - PSRAM (CRAM) 383
Figure 173. NAND/PC Card controller timing for common memory access 394
Figure 174. Access to non ‘CE don’t care’ NAND-Flash 395
Figure 175. SDIO “no response” and “no data” operations 404
Figure 176. SDIO (multiple) block read operation 404
Figure 177. SDIO (multiple) block write operation 405
Figure 178. SDIO sequential read operation 405
Figure 179. SDIO sequential write operation 405
Figure 180. SDIO block diagram 406
Figure 181. SDIO adapter 407
Figure 182. Control unit 408
Figure 183. SDIO adapter command path 409
Figure 184. Command path state machine (CPSM) 410
Figure 185. SDIO command transfer 411
Figure 186. Data path 413
Figure 187. Data path state machine (DPSM) 414
Figure 188. USB peripheral block diagram 459
Figure 189. Packet buffer areas with examples of buffer description table locations 464
Figure 190. CAN network topology 490
Figure 191. CAN general block diagram 491
Figure 192. 491
Figure 193. bxCAN operating modes. 492
Figure 194. bxCAN in silent mode 494
Figure 195. bxCAN in loop back mode 494
Figure 196. bxCAN in combined mode 495
Figure 197. Transmit mailbox states 496
Figure 198. Receive FIFO states 497
Figure 199. Filter bank scale configuration - register organization 500
Figure 200. Example of filter numbering 501
Figure 201. Filtering mechanism - example. 502
Figure 202. CAN error state diagram. 503

Figure 203. Bit timing . . . 505
Figure 204. CAN frames . . . 506
Figure 205. Event flags and interrupt generation . . . 507
Figure 206. SPI block diagram . . . 534
Figure 207. Single master/ single slave application . . . 535
Figure 208. Hardware/software slave select management . . . 535
Figure 209. Data clock timing diagram . . . 537
Figure 210. I^2S block diagram . . . 544
Figure 211. I^2S Phillips protocol waveforms (16/32-bit full accuracy, CPOL = 0) . . . 546
Figure 212. I^2S Phillips standard waveforms (24-bit frame with CPOL = 0) . . . 546
Figure 213. Transmitting 0x8EAA33 . . . 546
Figure 214. Receiving 0x8EAA33 . . . 547
Figure 215. I^2S Phillips standard (16-bit extended to 32-bit packet frame with CPOL = 0) . . . 547
Figure 216. Example . . . 547
Figure 217. MSB Justified 16-bit or 32-bit full-accuracy length with CPOL = 0 . . . 548
Figure 218. MSB Justified 24-bit frame length with CPOL = 0 . . . 548
Figure 219. MSB Justified 16-bit extended to 32-bit packet frame with CPOL = 0 . . . 548
Figure 220. LSB justified 16-bit or 32-bit full-accuracy with CPOL = 0 . . . 549
Figure 221. LSB Justified 24-bit frame length with CPOL = 0 . . . 549
Figure 222. Operations required to transmit 0x3478AE . . . 549
Figure 223. Operations required to receive 0x3478AE . . . 550
Figure 224. LSB Justified 16-bit extended to 32-bit packet frame with CPOL = 0 . . . 550
Figure 225. Example . . . 550
Figure 226. PCM standard waveforms (16-bit) . . . 551
Figure 227. PCM standard waveforms (16-bit extended to 32-bit packet frame) . . . 551
Figure 228. Audio sampling frequency definition . . . 552
Figure 229. I^2S clock generator architecture . . . 552
Figure 230. I2C bus protocol . . . 570
Figure 231. I2C block diagram . . . 571
Figure 232. Transfer sequence diagram for slave transmitter . . . 572
Figure 233. Transfer sequence diagram for slave receiver . . . 573
Figure 234. Transfer sequence diagram for master transmitter . . . 576
Figure 235. Transfer sequence diagram for master receiver . . . 577
Figure 236. I2C interrupt mapping diagram . . . 584
Figure 237. USART block diagram . . . 600
Figure 238. Word length programming . . . 601
Figure 239. Configurable stop bits . . . 603
Figure 240. Start bit detection . . . 605
Figure 241. Data sampling for noise detection . . . 607
Figure 242. Mute mode using Idle line detection . . . 610
Figure 243. Mute mode using Address mark detection . . . 611
Figure 244. Break detection in LIN mode (11-bit break length - LBDL bit is set) . . . 613
Figure 245. Break detection in LIN mode vs. Framing error detection . . . 614
Figure 246. USART example of synchronous transmission . . . 615
Figure 247. USART data clock timing diagram (M=0) . . . 615
Figure 248. USART data clock timing diagram (M=1) . . . 616
Figure 249. RX data setup/hold time . . . 616
Figure 250. ISO 7816-3 asynchronous protocol . . . 617
Figure 251. Parity error detection using the 1.5 stop bits . . . 618
Figure 252. IrDA SIR ENDEC- block diagram . . . 620
Figure 253. IrDA data modulation (3/16) -Normal Mode . . . 620
Figure 254. Hardware flow control between 2 USART . . . 622

Figure 255. RTS flow control 622
Figure 256. CTS flow control 623
Figure 257. USART interrupt mapping diagram 624
Figure 258. Block diagram of STM32F10xxx-level and Cortex-M3-level debug support. 641
Figure 259. SWJ debug port 642
Figure 260. JTAG TAP connections 646
Figure 261. TPIU block diagram 661

1 Documentation conventions

1.1 List of abbreviations for registers

The following abbreviations are used in register descriptions:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
write-only (w)	Software can only write to this bit. Reading the bit returns the reset value.
read/clear (rc_w1)	Software can read as well as clear this bit by writing 1. Writing ‘0’ has no effect on the bit value.
read/clear (rc_w0)	Software can read as well as clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
toggle (t)	Software can only toggle this bit by writing ‘1’. Writing ‘0’ has no effect.
Reserved (Res.)	Reserved bit, must be kept at reset value.

1.2 Glossary

- **Low-density devices** are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.
- **Medium-density devices** are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.
- **High-density devices** are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

1.3 Peripheral availability

For peripheral availability and number across all STM32F10xxx sales types, please refer to the low-, medium- and high-density STM32F101xx and STM32F103xx datasheets, and to the low- and medium-density STM32F102xx datasheets.

2 Memory and bus architecture

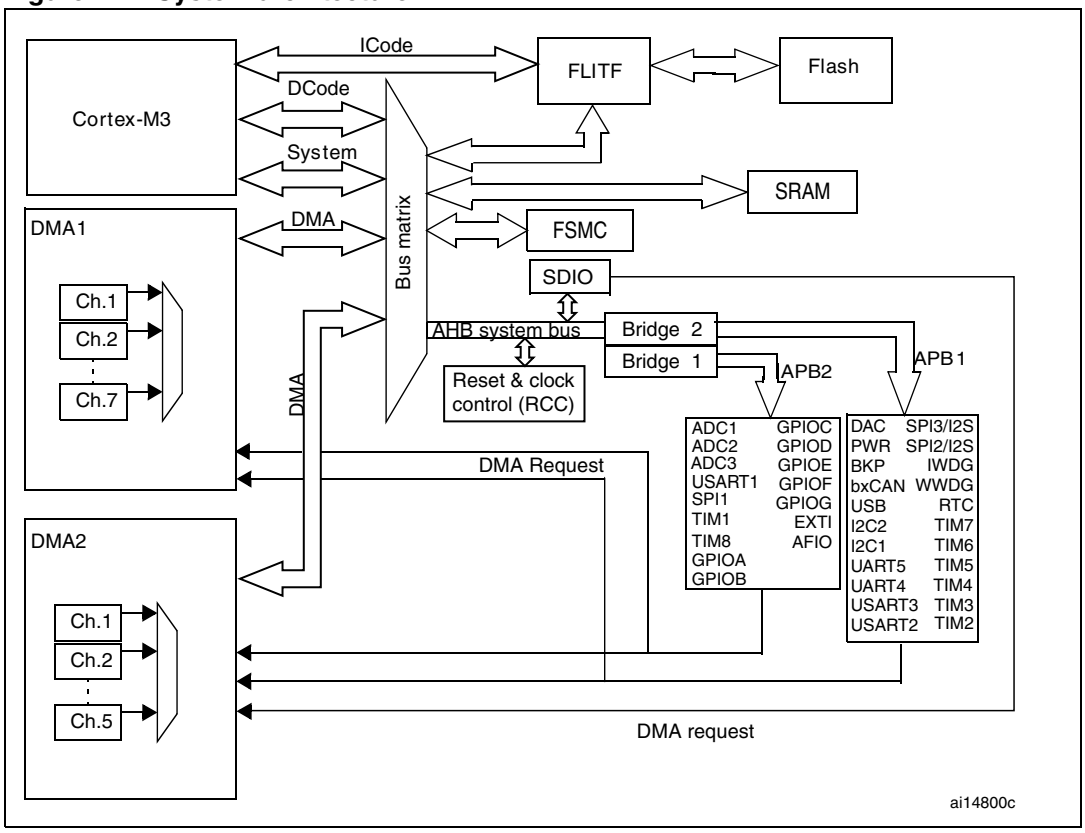
2.1 System architecture

The main system consists of:

- Four masters:
  - Cortex™-M3 core DCode bus (D-bus) and System bus (S-bus)
  - GP-DMA1 & 2 (general-purpose DMA)
- Four slaves:
  - Internal SRAM
  - Internal Flash memory
  - FSMC
  - AHB to APB bridges (AHB2APBx), which connect all the APB peripherals

These are interconnected using a multilayer AHB bus architecture as shown in *Figure 1*:

Figure 1. System architecture



ICode bus

This bus connects the Instruction bus of the Cortex™-M3 core to the Flash memory instruction interface. Prefetching is performed on this bus.

DCode bus

This bus connects the DCode bus (literal load and debug access) of the Cortex™-M3 core to the Flash memory Data interface.

System bus

This bus connects the system bus of the Cortex™-M3 core (peripherals bus) to a BusMatrix which manages the arbitration between the core and the DMA.

DMA bus

This bus connects the AHB master interface of the DMA to the BusMatrix which manages the access of CPU DCode and DMA to SRAM, Flash memory and peripherals.

BusMatrix

The BusMatrix manages the access arbitration between the core system bus and the DMA master bus. The arbitration uses a Round Robin algorithm. The BusMatrix is composed of four masters (CPU DCode, System bus, DMA1 bus and DMA2 bus) and four slaves (FLITF, SRAM, FSMC and AHB2APB bridges).

AHB peripherals are connected on system bus through a BusMatrix to allow DMA access.

AHB/APB bridges (APB)

The two AHB/APB bridges provide full synchronous connections between the AHB and the 2 APB buses. APB1 is limited to 36 MHz, APB2 operates at full speed (up to 72 MHz depending on the device).

Refer to *Table 1 on page 35* for the address mapping of the peripherals connected to each bridge.

After each device reset, all peripheral clocks are disabled (except for the SRAM and FLITF). Before using a peripheral you have to enable its clock in the RCC_AHBENR, RCC_APB2ENR or RCC_APB1ENR register.

Note: When a 16- or 8-bit access is performed on an APB register, the access is transformed into a 32-bit access: the bridge duplicates the 16- or 8-bit data to feed the 32-bit vector.

2.2 Memory organization

Program memory, data memory, registers and I/O ports are organized within the same linear 4-Gbyte address space.

The bytes are coded in memory in Little Endian format. The lowest numbered byte in a word is considered the word's least significant byte and the highest numbered byte the most significant.

For the detailed mapping of peripheral registers, please refer to the related chapters.

The addressable memory space is divided into 8 main blocks, each of 512 MB.

All the memory areas that are not allocated to on-chip memories and peripherals are considered "Reserved"). Refer to the Memory map figure in the corresponding product datasheet.

2.3 Memory map

See the datasheet corresponding to your device for a comprehensive diagram of the memory map. *Table 1* gives the boundary addresses of the peripherals available in all STM32F10xxx devices.

Table 1. Register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x4002 3400 - 0x4002 3FFF	Reserved	AHB	
0x4002 3000 - 0x4002 33FF	CRC		*Section 3.4.4 on page 44*
0x4002 2000 - 0x4002 23FF	Flash memory interface		
0x4002 1400 - 0x4002 1FFF	Reserved		
0x4002 1000 - 0x4002 13FF	Reset and clock control RCC		*Section 6.3.11 on page 92*
0x4002 0800 - 0x4002 0FFF	Reserved		
0x4002 0400 - 0x4002 07FF	DMA2		*Section 9.4.7 on page 141*
0x4002 0000 - 0x4002 03FF	DMA1		*Section 9.4.7 on page 141*
0x4001 8400 - 0x4001 7FFF	Reserved		
0x4001 8000 - 0x4001 83FF	SDIO		*Section 19.9.16 on page 456*
0x4001 4000 - 0x4001 7FFF	Reserved	APB2	
0x4001 3C00 - 0x4001 3FFF	ADC3		*Section 10.12.15 on page 176*
0x4001 3800 - 0x4001 3BFF	USART1		*Section 24.6.8 on page 636*
0x4001 3400 - 0x4001 37FF	TIM8 timer		*Section 12.4.21 on page 264*
0x4001 3000 - 0x4001 33FF	SPI1		*Section 22.5 on page 557*
0x4001 2C00 - 0x4001 2FFF	TIM1 timer		*Section 12.4.21 on page 264*
0x4001 2800 - 0x4001 2BFF	ADC2		*Section 10.12.15 on page 176*
0x4001 2400 - 0x4001 27FF	ADC1		*Section 10.12.15 on page 176*
0x4001 2000 - 0x4001 23FF	GPIO Port G		*Section 7.5 on page 115*
0x4001 1C00 - 0x4001 1FFF	GPIO Port F		*Section 7.5 on page 115*
0x4001 1800 - 0x4001 1BFF	GPIO Port E		*Section 7.5 on page 115*
0x4001 1400 - 0x4001 17FF	GPIO Port D		*Section 7.5 on page 115*
0x4001 1000 - 0x4001 13FF	GPIO Port C		*Section 7.5 on page 115*
0x4001 0C00 - 0x4001 0FFF	GPIO Port B		*Section 7.5 on page 115*
0x4001 0800 - 0x4001 0BFF	GPIO Port A		*Section 7.5 on page 115*
0x4001 0400 - 0x4001 07FF	EXTI		*Section 8.3.7 on page 127*
0x4001 0000 - 0x4001 03FF	AFIO		*Section 7.5 on page 115*

Table 1. Register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4000 7800 - 0x4000 FFFF	Reserved	APB1	
0x4000 7400 - 0x4000 77FF	DAC		*Section 11.5.14 on page 198*
0x4000 7000 - 0x4000 73FF	Power control PWR		*Section 4.4.3 on page 57*
0x4000 6C00 - 0x4000 6FFF	Backup registers (BKP)		*Section 5.4.5 on page 63*
0x4000 6800 - 0x4000 6BFF	Reserved		
0x4000 6400 - 0x4000 67FF	bxCAN		*Section 21.6.5 on page 527*
0x4000 6000 - 0x4000 63FF	Shared USB/CAN SRAM 512 bytes		
0x4000 5C00 - 0x4000 5FFF	USB device FS registers		*Section 20.5.4 on page 487*
0x4000 5800 - 0x4000 5BFF	I2C2		*Section 23.6.10 on page 596*
0x4000 5400 - 0x4000 57FF	I2C1		*Section 23.6.10 on page 596*
0x4000 5000 - 0x4000 53FF	UART5		*Section 24.6.8 on page 636*
0x4000 4C00 - 0x4000 4FFF	UART4		*Section 24.6.8 on page 636*
0x4000 4800 - 0x4000 4BFF	USART3		*Section 24.6.8 on page 636*
0x4000 4400 - 0x4000 47FF	USART2		*Section 24.6.8 on page 636*
0x4000 4000 - 0x4000 3FFF	Reserved		
0x4000 3C00 - 0x4000 3FFF	SPI3/I2S		*Section 22.5 on page 557*
0x4000 3800 - 0x4000 3BFF	SPI2/I2S		*Section 22.5 on page 557*
0x4000 3400 - 0x4000 37FF	Reserved		
0x4000 3000 - 0x4000 33FF	Independent watchdog (IWDG)		*Section 16.4.5 on page 351*
0x4000 2C00 - 0x4000 2FFF	Window watchdog (WWDG)		*Section 17.6.4 on page 356*
0x4000 2800 - 0x4000 2BFF	RTC		*Section 15.4.7 on page 345*
0x4000 1800 - 0x4000 27FF	Reserved		
0x4000 1400 - 0x4000 17FF	TIM7 timer		*Section 14.4.9 on page 333*
0x4000 1000 - 0x4000 13FF	TIM6 timer		*Section 14.4.9 on page 333*
0x4000 0C00 - 0x4000 0FFF	TIM5 timer		*Section 13.4.19 on page 320*
0x4000 0800 - 0x4000 0BFF	TIM4 timer		*Section 13.4.19 on page 320*
0x4000 0400 - 0x4000 07FF	TIM3 timer		*Section 13.4.19 on page 320*
0x4000 0000 - 0x4000 03FF	TIM2 timer		*Section 13.4.19 on page 320*

2.3.1 Embedded SRAM

The STM32F10xxx features 64 Kbytes of static SRAM. It can be accessed as bytes, half-words (16 bits) or full words (32 bits). The SRAM start address is 0x2000 0000.

2.3.2 Bit banding

The Cortex™-M3 memory map includes two bit-band regions. These regions map each word in an alias region of memory to a bit in a bit-band region of memory. Writing to a word

in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.

In the STM32F10xxx both peripheral registers and SRAM are mapped in a bit-band region. This allows single bit-band write and read operations to be performed.

A mapping formula shows how to reference each word in the alias region to a corresponding bit in the bit-band region. The mapping formula is:

$$bit_word_addr = bit_band_base + (byte_offset \times 32) + (bit_number \times 4)$$

where:

bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.

bit_band_base is the starting address of the alias region

byte_offset is the number of the byte in the bit-band region that contains the targeted bit

bit_number is the bit position (0-7) of the targeted bit.

Example:

The following example shows how to map bit 2 of the byte located at SRAM address 0x20000300 in the alias region:

0x22006008 = 0x22000000 + (0x300*32) + (2*4).

Writing to address 0x22006008 has the same effect as a read-modify-write operation on bit 2 of the byte at SRAM address 0x20000300.

Reading address 0x22006008 returns the value (0x01 or 0x00) of bit 2 of the byte at SRAM address 0x20000300 (0x01: bit set; 0x00: bit reset).

For more information on Bit-Banding, please refer to the *Cortex™-M3 Technical Reference Manual*.

2.3.3 Embedded Flash memory

The high-performance Flash memory module has the following key features:

- Density of up to 512 Kbytes
- Memory organization: the Flash memory is organized as a main block and an information block:
  - Main memory block of size:

    up to 4 Kb × 64 bits divided into 32 pages of 1 Kbyte each for low-density devices (see *Table 2*)

    up to 16 Kb × 64 bits divided into 128 pages of 1 Kbyte each for medium-density devices (see *Table 3*)

    up to 64 Kb × 64 bits divided into 256 pages of 2 Kbytes each (see *Table 4*) for high-density devices
  - Information block of size 258 × 64 bits. The information block is divided into 2 pages of 2 Kbytes and 16 bytes, respectively (see *Table 3*).

The Flash memory interface (FLITF) features:

- Read interface with prefetch buffer (2x64-bit words)
- Option byte Loader
- Flash Program / Erase operation
- Read / Write protection

Table 2. Flash module organization (low-density devices)

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 Kbyte
	Page 1	0x0800 0400 - 0x0800 07FF	1 Kbyte
	Page 2	0x0800 0800 - 0x0800 0BFF	1 Kbyte
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 Kbyte
	Page 4	0x0800 1000 - 0x0800 13FF	1 Kbyte
	. . .	. . .	. . .
	Page 31	0x0800 7C00 - 0x0800 7FFF	1 Kbyte
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

Table 3. Flash module organization (medium-density devices)

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 Kbyte
	Page 1	0x0800 0400 - 0x0800 07FF	1 Kbyte
	Page 2	0x0800 0800 - 0x0800 0BFF	1 Kbyte
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 Kbyte
	Page 4	0x0800 1000 - 0x0800 13FF	1 Kbyte
	. . .	. . .	. . .
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 Kbyte
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16

Table 3. Flash module organization (medium-density devices)

Block	Name	Base addresses	Size (bytes)
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

Table 4. Flash module organization (high-density devices)

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 07FF	2 Kbytes
	Page 1	0x0800 0800 - 0x0800 0FFF	2 Kbytes
	Page 2	0x0800 1000 - 0x0800 17FF	2 Kbytes
	Page 3	0x0800 1800 - 0x0800 1FFF	2 Kbytes
	. . .	. . .	. . .
	Page 255	0x0807 F800 - 0x0807 FFFF	2 Kbytes
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

Note: For further information on the Flash memory interface registers, please refer to the STM32F10xxx Flash programming manual.

Reading Flash memory

Flash memory instructions and data access are performed through the AHB bus. The prefetch block is used for instruction fetches through the ICode bus. Arbitration is performed in the Flash memory interface, and priority is given to data access on the DCode bus.

Read accesses can be performed with the following configuration options:

- Latency: number of wait states for a read operation programmed on-the-fly
- Prefetch buffer (2 x 64-bit blocks): it is enabled after reset; a whole block can be replaced with a single read from the Flash memory as the size of the block matches the bandwidth of the Flash memory. Thanks to the prefetch buffer, faster CPU execution is possible as the CPU fetches one word at a time with the next word readily available in the prefetch buffer
- Half cycle: for power optimization

Note: 1 These options should be used in accordance with the Flash memory access time. The wait states represent the ratio of the SYSCLK (system clock) period to the Flash memory access time:

zero wait state, if 0 < SYSCLK ≤ 24 MHz
one wait state, if 24 MHz < SYSCLK ≤ 48 MHz
two wait states, if 48 MHz < SYSCLK ≤ 72 MHz

2 Half cycle configuration is not available in combination with a prescaler on the AHB. The system clock (SYSCLK) should be equal to the HCLK clock. This feature can therefore be used only with a low-frequency clock of 8 MHz or less. It can be generated from the HSI or the HSE but not from the PLL.

3 The prefetch buffer must be kept on when using a prescaler different from 1 on the AHB clock.

4 The prefetch buffer must be switched on/off only when SYSCLK is lower than 24 MHz. The prefetch buffer is usually switched on/off during the initialization routine, while the microcontroller is running on the internal 8 MHz RC (HSI) oscillator.

5 Using DMA: DMA accesses Flash memory on the DCode bus and has priority over ICode instructions. The DMA provides one free cycle after each transfer. Some instructions can be performed together with DMA transfer.

Programming and erasing Flash memory

The Flash memory can be programmed 16 bits (half words) at a time.

The Flash memory erase operation can be performed at page level or on the whole Flash area (mass-erase). The mass-erase does not affect the information blocks.

To ensure that there is no over-programming, the Flash Programming and Erase Controller blocks are clocked by a fixed clock.

The End of write operation (programming or erasing) can trigger an interrupt. This interrupt can be used to exit from WFI mode, only if the FLITF clock is enabled. Otherwise, the interrupt is served only after an exit from WFI.

Note: For further information on Flash memory operations and register configurations, please refer to the STM32F10xxx Flash programming manual.

2.4 Boot configuration

In the STM32F10xxx, 3 different boot modes can be selected through BOOT[1:0] pins as shown in *Table 5*.

Table 5. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as boot space
0	1	System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

The values on the BOOT pins are latched on the 4th rising edge of SYSCLK after a Reset. It is up to the user to set the BOOT1 and BOOT0 pins after Reset to select the required boot mode.

The BOOT pins are also re-sampled when exiting from Standby mode. Consequently they must be kept in the required Boot mode configuration in Standby mode. After this startup delay has elapsed, the CPU fetches the top-of-stack value from address 0x0000 0000, then starts code execution from the boot memory starting from 0x0000 0004.

Due to its fixed memory map, the code area starts from address 0x0000 0000 (accessed through the ICode/DCode buses) while the data area (SRAM) starts from address 0x2000 0000 (accessed through the system bus). The Cortex-M3 CPU always fetches the reset vector on the ICode bus, which implies to have the boot space available only in the code area (typically, Flash memory). STM32F10xxx microcontrollers implement a special mechanism to be able to boot also from SRAM and not only from main Flash memory and System memory.

Depending on the selected boot mode main Flash memory, System memory or SRAM is accessible as follows:

- Boot from main Flash memory: the main Flash memory is aliased in the boot memory space (0x0000 0000), but still accessible from its original memory space (0x800 0000). In other words, the Flash memory contents can be accessed starting from address 0x0000 0000 or 0x800 0000.
- Boot from System memory: the System memory is aliased in the boot memory space (0x0000 0000), but still accessible from its original memory space (0x1FFF F000).
- Boot from the embedded SRAM: SRAM is accessible only at address 0x2000 0000.

Note: When booting from SRAM, in the application initialization code, you have to relocate the vector table in SRAM using the NVIC exception table and offset register.

Embedded boot loader

The embedded boot loader is used to reprogram the Flash memory using the USART1 serial interface. This program is located in the System memory and is programmed by ST during production. For further details please refer to AN2606.

3 CRC calculation unit

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

3.1 CRC introduction

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from a 32-bit data word and a fixed generator polynomial.

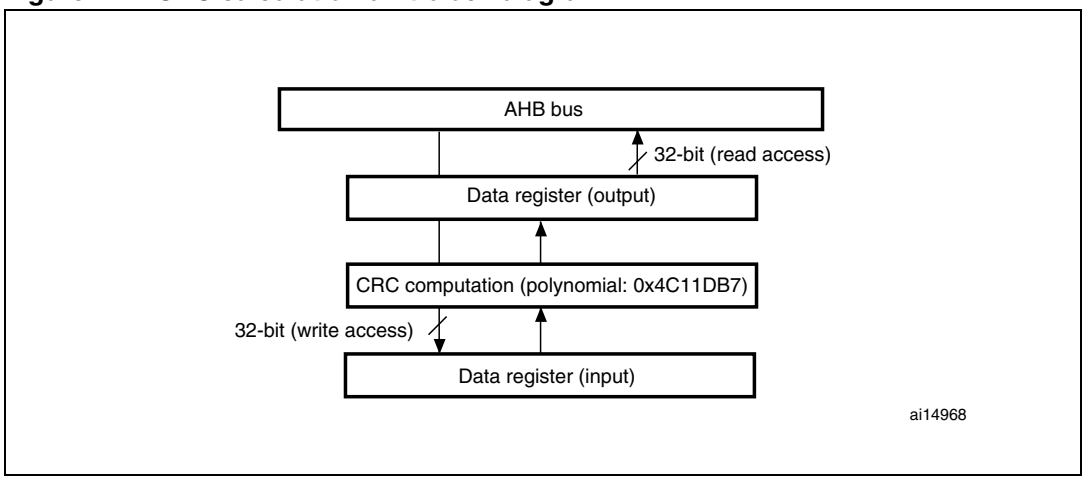
Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the EN/IEC 60335-1 standard, they offer a means of verifying the Flash memory integrity. The CRC calculation unit helps compute a signature of the software during runtime, to be compared with a reference signature generated at link-time and stored at a given memory location.

3.2 CRC main features

- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7
  - $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$
- Single input/output 32-bit data register
- CRC computation done in 4 AHB clock cycles (HCLK)
- General-purpose 8-bit register (can be used for temporary storage)

The block diagram is shown in *Figure 2*.

Figure 2. CRC calculation unit block diagram



3.3 CRC functional description

The CRC calculation unit mainly consists of a single 32-bit data register, which:

- is used as an input register to enter new data in the CRC calculator (when writing into the register)
- holds the result of the previous CRC calculation (when reading the register)

Each write operation into the data register creates a combination of the previous CRC value and the new one (CRC computation is done on the whole 32-bit data word, and not byte per byte).

The CPU is stalled during the computation, thus allowing back-to-back write accesses or consecutive write and read accesses, without having to insert software wait cycles.

The CRC calculator can be reset to FFFF FFFFh with the RESET control bit in the CRC_CR register. This operation does not affect the contents of the CRC_IDR register.

3.4 CRC registers

The CRC calculation unit contains two data registers and a control register.

3.4.1 Data register (CRC_DR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DR [31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR [15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **Data register bits**

Used as an input register when writing new data into the CRC calculator.
Holds the previous CRC calculation result when it is read.

3.4.2 Independent data register (CRC_IDR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								IDR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 **Reserved**

Bits 7:0 **General-purpose 8-bit data register bits**

Can be used as a temporary storage location for one byte.
This register is not affected by CRC resets generated by the RESET bit in the CRC_CR register.

3.4.3 Control register (CRC_CR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															RESET
															w

Bits 31:1 **Reserved**

RESET bit

Bit 0 Resets the CRC calculation unit and sets the data register to FFFF FFFFh.
This bit can only be set, it is automatically cleared by hardware.

3.4.4 CRC register map

The following table provides the CRC register map and reset values.

Table 6. CRC calculation unit register map and reset values

Offset	Register	31-24	23-16	15-8	7	6	5	4	3	2	1	0
0x00	**CRC_DR** Reset value	Data register 0xFFFF FFFF										
0x04	**CRC_IDR** Reset value	Reserved			Independent data register 0x00							
0x08	**CRC_CR** Reset value	Reserved								Reserved 0		RESET 0

4 Power control (PWR)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

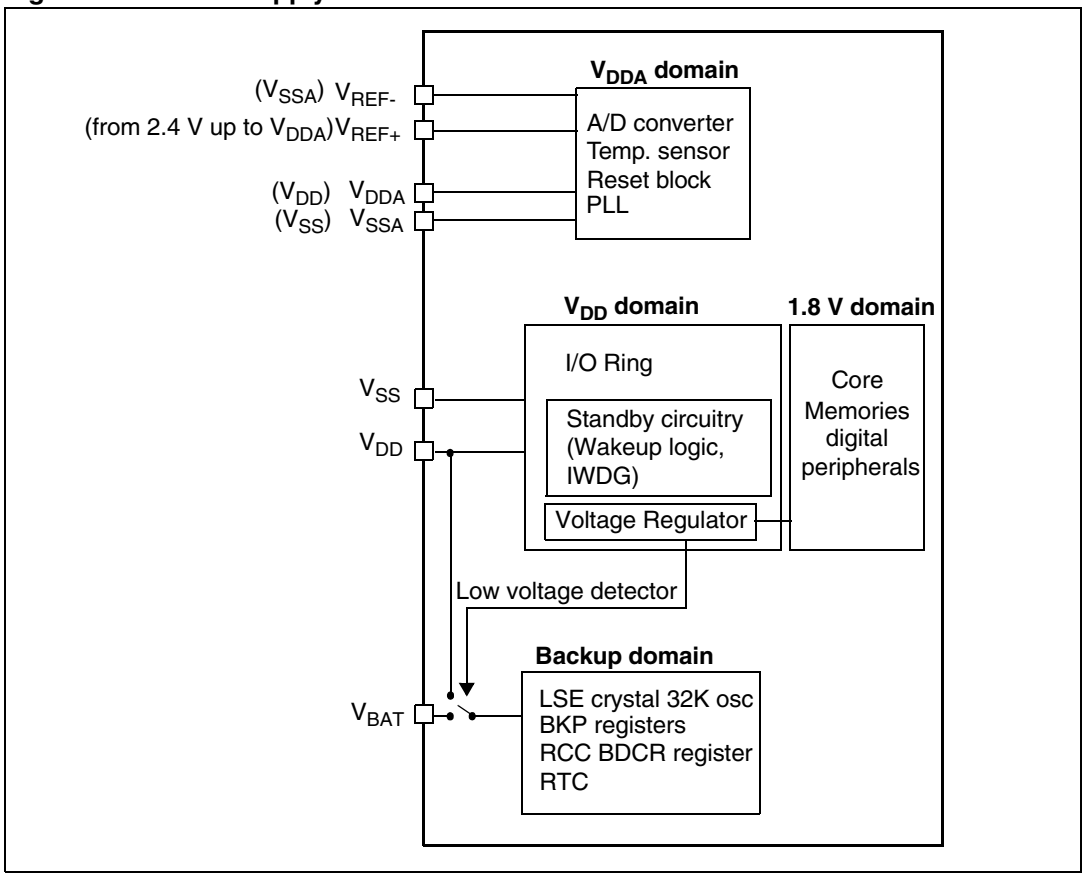
This section applies to the whole STM32F10xxx family, unless otherwise specified.

4.1 Power supplies

The device requires a 2.0-to-3.6 V operating voltage supply (V_{DD}). An embedded regulator is used to supply the internal 1.8 V digital power.

The real-time clock (RTC) and backup registers can be powered from the V_{BAT} voltage when the main V_{DD} supply is powered off.

Figure 3. Power supply overview



Note: 1 V_{DDA} and V_{SSA} must be connected to V_{DD} and V_{SS}, respectively.

4.1.1 Independent A/D converter supply and reference voltage

To improve conversion accuracy, the ADC has an independent power supply which can be separately filtered and shielded from noise on the PCB.

- The ADC voltage supply input is available on a separate V_{DDA} pin.
- An isolated supply ground connection is provided on pin V_{SSA}.

When available (according to package), V_{REF-} must be tied to V_{SSA}.

On 100-pin and 144- pin packages

To ensure a better accuracy on low voltage inputs, the user can connect a separate external reference voltage ADC input on V_{REF+} and V_{REF-}. The voltage on V_{REF+} can range from 2.4 V to V_{DDA}.

On 64-pin packages

The V_{REF+} and V_{REF-} pins are not available, they are internally connected to the ADC voltage supply (V_{DDA}) and ground (V_{SSA}).

4.1.2 Battery backup domain

To retain the content of the Backup registers and supply the RTC function when V_{DD} is turned off, V_{BAT} pin can be connected to an optional standby voltage supplied by a battery or by another source.

The V_{BAT} pin powers the RTC unit, the LSE oscillator and the PC13 to PC15 IOs, allowing the RTC to operate even when the main digital supply (V_{DD}) is turned off. The switch to the V_{BAT} supply is controlled by the Power Down Reset embedded in the Reset block.

Warning: **During $t_{RSTTEMPO}$ (temporization at V_{DD} startup) or after a PDR is detected, the power switch between V_{BAT} and V_{DD} remains connected to V_{BAT}.**
During the startup phase, if V_{DD} is established in less than $t_{RSTTEMPO}$ (Refer to the datasheet for the value of $t_{RSTTEMPO}$) and $V_{DD} > V_{BAT} + 0.6$ V, a current may be injected into V_{BAT} through an internal diode connected between V_{DD} and the power switch (V_{BAT}).
If the power supply/battery connected to the V_{BAT} pin cannot support this current injection, it is strongly recommended to connect an external low-drop diode between this power supply and the V_{BAT} pin.

If no external battery is used in the application, it is recommended to connect V_{BAT} externally to V_{DD} through a 100 nF external ceramic capacitor (for more details refer to AN2586).

When the backup domain is supplied by V_{DD} (analog switch connected to V_{DD}), the following functions are available:

- PC14 and PC15 can be used as either GPIO or LSE pins
- PC13 can be used as GPIO, TAMPER pin, RTC Calibration Clock, RTC Alarm or second output (refer to *Section 5: Backup registers (BKP) on page 58*)

Note: Due to the fact that the switch only sinks a limited amount of current (3 mA), the use of GPIOs PC13 to PC15 is restricted: only one I/O at a time can be used as an output, the speed has to be limited to 2 MHz with a maximum load of 30 pF and these IOs must not be used as a current source (e.g. to drive an LED).

When the backup domain is supplied by V_{BAT} (analog switch connected to V_{BAT} because V_{DD} is not present), the following functions are available:

- PC14 and PC15 can be used as LSE pins only
- PC13 can be used as TAMPER pin, RTC Alarm or Second output (refer to section *Section 5.4.2: RTC clock calibration register (BKP_RTCCR) on page 60*).

4.1.3 Voltage regulator

The voltage regulator is always enabled after Reset. It works in three different modes depending on the application modes.

- In Run mode, the regulator supplies full power to the 1.8 V domain (core, memories and digital peripherals).
- In Stop mode the regulator supplies low-power to the 1.8 V domain, preserving contents of registers and SRAM
- In Standby Mode, the regulator is powered off. The contents of the registers and SRAM are lost except for the Standby circuitry and the Backup Domain.

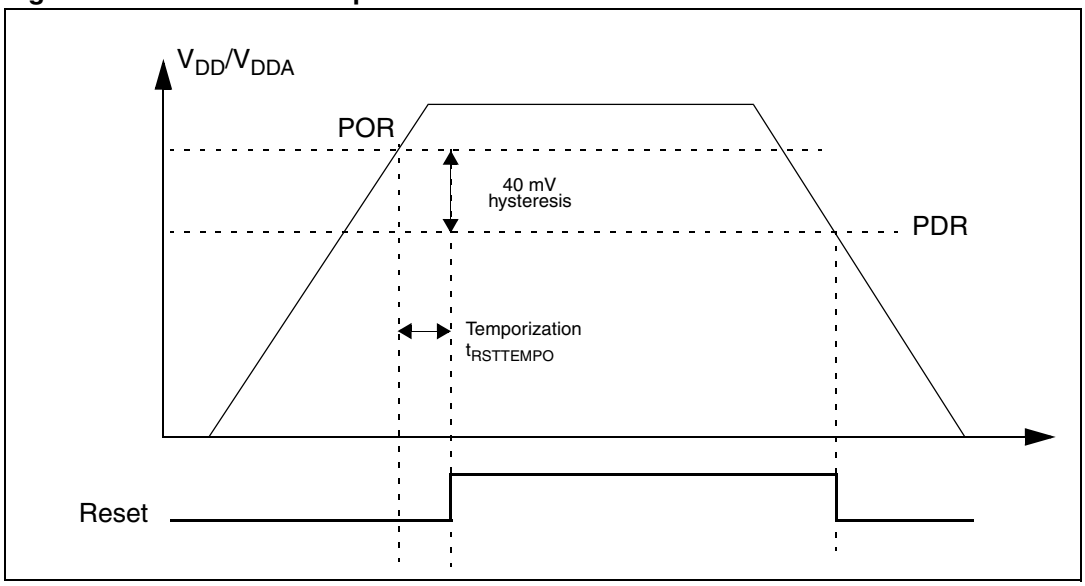
4.2 Power supply supervisor

4.2.1 Power on reset (POR)/power down reset (PDR)

The device has an integrated POR/PDR circuitry that allows proper operation starting from/down to 2 V.

The device remains in Reset mode when V_{DD}/V_{DDA} is below a specified threshold, $V_{POR/PDR}$, without the need for an external reset circuit. For more details concerning the power on/power down reset threshold, refer to the electrical characteristics of the datasheet.

Figure 4. Power on reset/power down reset waveform



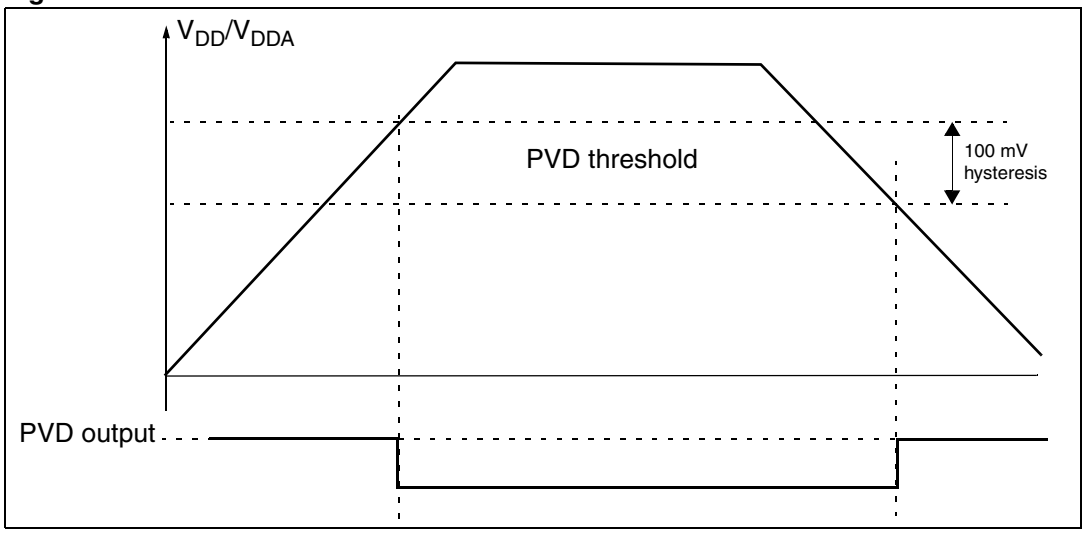
4.2.2 Programmable voltage detector (PVD)

You can use the PVD to monitor the V_{DD}/V_{DDA} power supply by comparing it to a threshold selected by the PLS[2:0] bits in the *Power control register (PWR_CR)*.

The PVD is enabled by setting the PVDE bit.

A PVDO flag is available, in the *Power control/status register (PWR_CSR)*, to indicate if V_{DD}/V_{DDA} is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled through the EXTI registers. The PVD output interrupt can be generated when V_{DD}/V_{DDA} drops below the PVD threshold and/or when V_{DD}/V_{DDA} rises above the PVD threshold depending on EXTI line16 rising/falling edge configuration. As an example the service routine could perform emergency shutdown tasks.

Figure 5. PVD thresholds



4.3 Low-power modes

By default, the microcontroller is in Run mode after a system or a power Reset. In Run mode the CPU is clocked by HCLK and the program code is executed. Several low-power modes are available to save power when the CPU does not need to be kept running, for example when waiting for an external event. It is up to the user to select the mode that gives the best compromise between low-power consumption, short startup time and available wakeup sources.

The STM32F10xxx devices feature three low-power modes:

- Sleep mode (Cortex-M3 core stopped, peripherals kept running)
- Stop mode (all clocks are stopped)
- Standby mode (1.8V domain powered-off)

In addition, the power consumption in Run mode can be reduce by one of the following means:

- Slowing down the system clocks
- Gating the clocks to the APB and AHB peripherals when they are unused.

Table 7. Low-power mode summary

Mode name	Entry	wakeup	Effect on 1.8V domain clocks	Effect on V_{DD} domain clocks	Voltage regulator
Sleep (Sleep now or Sleep-on - exit)	WFI	Any interrupt	CPU CLK OFF no effect on other clocks or analog clock sources	None	ON
	WFE	Wakeup event			
Stop	PDDS and LPDS bits + SLEEPDEEP bit + WFI or WFE	Any EXTI line (configured in the EXTI registers)	All 1.8V domain clocks OFF	HSI and HSE oscillators OFF	ON or in low-power mode (depends on *Power control register (PWR_CR)*)
Standby	PDDS bit + SLEEPDEEP bit + WFI or WFE	WKUP pin rising edge, RTC alarm, external reset in NRST pin, IWDG reset			OFF

4.3.1 Slowing down system clocks

In Run mode the speed of the system clocks (SYSCLK, HCLK, PCLK1, PCLK2) can be reduced by programming the prescaler registers. These prescalers can also be used to slow down peripherals before entering Sleep mode.

For more details refer to *Section 6.3.2: Clock configuration register (RCC_CFGR)*.

4.3.2 Peripheral clock gating

In Run mode, the HCLK and PCLKx for individual peripherals and memories can be stopped at any time to reduce power consumption.

To further reduce power consumption in Sleep mode the peripheral clocks can be disabled prior to executing the WFI or WFE instructions.

Peripheral clock gating is controlled by the *AHB peripheral clock enable register (RCC_AHBENR)*, *APB1 peripheral clock enable register (RCC_APB1ENR)* and *APB2 peripheral clock enable register (RCC_APB2ENR)*.

4.3.3 Sleep mode

Entering Sleep mode

The Sleep mode is entered by executing the WFI (Wait For Interrupt) or WFE (Wait for Event) instructions. Two options are available to select the Sleep mode entry mechanism, depending on the SLEEPONEXIT bit in the Cortex-M3 System Control register:

- Sleep-now: if the SLEEPONEXIT bit is cleared, the MCU enters Sleep mode as soon as WFI or WFE instruction is executed.
- Sleep-on-exit: if the SLEEPONEXIT bit is set, the MCU enters Sleep mode as soon as it exits the lowest priority ISR.

Refer to *Table 8* and *Table 9* for details on how to enter Sleep mode.

Exiting Sleep mode

If the WFI instruction is used to enter Sleep mode, any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

If the WFE instruction is used to enter Sleep mode, the MCU exits Sleep mode as soon as an event occurs. The wakeup event can be generated either by:

- enabling an interrupt in the peripheral control register but not in the NVIC, and enabling the SEVONPEND bit in the Cortex-M3 System Control register. When the MCU resumes from WFE, the peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.
- or configuring an external or internal EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bit corresponding to the event line is not set.

This mode offers the lowest wakeup time as no time is wasted in interrupt entry/exit.

Refer to *Table 8* and *Table 9* for more details on how to exit Sleep mode.

Table 8. Sleep-now

Sleep-now mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 0 Refer to the Cortex™-M3 System Control register.
Mode exit	If WFI was used for entry: Interrupt: Refer to *Table 36: Vector table for other STM32F10xxx devices* If WFE was used for entry Wakeup event: Refer to *Section 8.2.3: Wakeup event management*
Wakeup latency	None

Table 9. Sleep-on-exit

Sleep-on-exit	Description
Mode entry	WFI (wait for interrupt) while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 1 Refer to the Cortex™-M3 System Control register.
Mode exit	Interrupt: refer to *Table 36: Vector table for other STM32F10xxx devices*.
Wakeup latency	None

4.3.4 Stop mode

The Stop mode is based on the Cortex-M3 deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the 1.8 V domain are stopped, the PLL, the HSI and the HSE RC oscillators are disabled. SRAM and register contents are preserved.

Entering Stop mode

Refer to *Table 10* for details on how to enter the Stop mode.

To further reduce power consumption in Stop mode, the internal voltage regulator can be put in low-power mode. This is configured by the LPDS bit of the *Power control register (PWR_CR)*.

If Flash memory programming is ongoing, the Stop mode entry is delayed until the memory access is finished.

If an access to the APB domain is ongoing, The Stop mode entry is delayed until the APB access is finished.

In Stop mode, the following features can be selected by programming individual control bits:

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a Reset. See *Section 16.3* in *Section 16: Independent watchdog (IWDG)*.
- real-time clock (RTC): this is configured by the RTCEN bit in the *Backup domain control register (RCC_BDCR)*
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the *Control/status register (RCC_CSR)*.
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the *Backup domain control register (RCC_BDCR)*.

The ADC or DAC can also consume power during the Stop mode, unless they are disabled before entering it. To disable them, the ADON bit in the ADC_CR2 register and the ENx bit in the DAC_CR register must both be written to 0.

Exiting Stop mode

Refer to *Table 10* for more details on how to exit Stop mode.

When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.

When the voltage regulator operates in low-power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

Table 10. Stop mode

Stop mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – Set SLEEPDEEP bit in Cortex™-M3 System Control register – Clear PDDS bit in Power Control register (PWR_CR) – Select the voltage regulator mode by configuring LPDS bit in PWR_CR **Note:** To enter Stop mode, all EXTI Line pending bits (in *Pending register (EXTI_PR)*) and RTC Alarm flag must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues.
Mode exit	If WFI was used for entry: Any EXTI Line configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC). Refer to *Table 36: Vector table for other STM32F10xxx devices on page 117*. If WFE was used for entry: Any EXTI Line configured in event mode. Refer to *Section 8.2.3: Wakeup event management on page 121*
Wakeup latency	HSI RC wakeup time + regulator wakeup time from Low-power mode

4.3.5 Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M3 deepsleep mode, with the voltage regulator disabled. The 1.8 V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also

switched off. SRAM and register contents are lost except for registers in the Backup domain and Standby circuitry (see *Figure 3*).

Entering Standby mode

Refer to *Table 11* for more details on how to enter Standby mode.

In Standby mode, the following features can be selected by programming individual control bits:

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a reset. See *Section 16.3* in *Section 16: Independent watchdog (IWDG)*.
- real-time clock (RTC): this is configured by the RTCEN bit in the Backup domain control register (RCC_BDCR)
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the Control/status register (RCC_CSR).
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the Backup domain control register (RCC_BDCR)

Exiting Standby mode

The microcontroller exits Standby mode when an external Reset (NRST pin), IWDG Reset, a rising edge on WKUP pin or an RTC alarm occurs. All registers are reset after wakeup from Standby except for *Power control/status register (PWR_CSR)*.

After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pins sampling, vector reset is fetched, etc.). The SBF status flag in the *Power control/status register (PWR_CSR)* indicates that the MCU was in Standby mode.

Refer to *Table 11* for more details on how to exit Standby mode.

Table 11. Standby mode

Standby mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – Set SLEEPDEEP in Cortex™-M3 System Control register – Set PDDS bit in Power Control register (PWR_CR) – Clear WUF bit in Power Control/Status register (PWR_CSR)
Mode exit	WKUP pin rising edge, RTC alarm, external Reset in NRST pin, IWDG Reset.
Wakeup latency	Regulator start up. Reset phase

I/O states in Standby mode

In Standby mode, all I/O pins are high impedance except:

- Reset pad (still available)
- TAMPER pin if configured for tamper or calibration out
- WKUP pin, if enabled

Debug mode

By default, the debug connection is lost if the application puts the MCU in Stop or Standby mode while the debug features are used. This is due to the fact that the Cortex™-M3 core is no longer clocked.

However, by setting some configuration bits in the DBGMCU_CR register, the software can be debugged even when using the low-power modes extensively. For more details, refer to *Section 26.15.1: Debug support for low-power modes*.

4.3.6 Auto-wakeup (AWU) from low-power mode

The RTC can be used to wakeup the MCU from low-power mode without depending on an external interrupt (Auto-wakeup mode). The RTC provides a programmable time base for waking up from Stop or Standby mode at regular intervals. For this purpose, two of the three alternative RTC clock sources can be selected by programming the RTCSEL[1:0] bits in the *Backup domain control register (RCC_BDCR)*:

- Low-power 32.768 kHz external crystal oscillator (LSE OSC).
  This clock source provides a precise time base with very low-power consumption (less than 1µA added consumption in typical conditions)
- Low-power internal RC Oscillator (LSI RC)
  This clock source has the advantage of saving the cost of the 32.768 kHz crystal. This internal RC Oscillator is designed to add minimum power consumption.

To wakeup from Stop mode with an RTC alarm event, it is necessary to:

- Configure the EXTI Line 17 to be sensitive to rising edge
- Configure the RTC to generate the RTC alarm

To wakeup from Standby mode, there is no need to configure the EXTI Line 17.

4.4 Power control registers

4.4.1 Power control register (PWR_CR)

Address offset: 0x00

Reset value: 0x0000 0000 (reset by wakeup from Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
Res.															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DBP	PLS[2:0]			PVDE	CSBF	CWUF	PDDS	LPDS
Res							rw	rw	rw	rw	rw	rc_w1	rc_w1	rw	rw

Bits 31:9 Reserved, always read as 0.

Bit 8 **DBP**: Disable backup domain write protection.

In reset state, the RTC and backup registers are protected against parasitic write access. This bit must be set to enable write access to these registers.

0: Access to RTC and Backup registers disabled

1: Access to RTC and Backup registers enabled

Bits 7:5 **PLS[2:0]:** PVD level selection.

These bits are written by software to select the voltage threshold detected by the Power Voltage Detector

000: 2.2V

001: 2.3V

010: 2.4V

011: 2.5V

100: 2.6V

101: 2.7V

110: 2.8V

111: 2.9V

Note: Refer to the electrical characteristics of the datasheet for more details.

Bit 4 **PVDE:** Power voltage detector enable.

This bit is set and cleared by software.

0: PVD disabled

1: PVD enabled

Bit 3 **CSBF**: Clear standby flag.

This bit is always read as 0.

0: No effect

1: Clear the SBF Standby Flag (write).

Bit 2 **CWUF:** Clear wakeup flag.

This bit is always read as 0.

0: No effect

1: Clear the WUF Wakeup Flag **after 2 System clock cycles**. (write)

Bit 1 **PDDS**: Power down deepsleep.

This bit is set and cleared by software. It works together with the LPDS bit.

0: Enter Stop mode when the CPU enters Deepsleep. The regulator status depends on the LPDS bit.

1: Enter Standby mode when the CPU enters Deepsleep.

Bit 0 **LPDS:** Low-power deepsleep.

This bit is set and cleared by software. It works together with the PDDS bit.

0: Voltage regulator on during Stop mode

1: Voltage regulator in low-power mode during Stop mode

4.4.2 Power control/status register (PWR_CSR)

Address offset: 0x04

Reset value: 0x0000 0000 (not reset by wakeup from Standby mode)

Additional APB cycles are needed to read this register versus a standard APB read.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
Res.															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							EWUP	Reserved					PVDO	SBF	WUF
Res.							rw	Res.					r	r	r

Bits 31:9 Reserved, always read as 0.

Bit 8 **EWUP**: Enable WKUP pin

This bit is set and cleared by software.

0: WKUP pin is used for general purpose I/O. An event on the WKUP pin does not wakeup the device from Standby mode.

1: WKUP pin is used for wakeup from Standby mode and forced in input pull down configuration (rising edge on WKUP pin wakes-up the system from Standby mode).

Note: This bit is reset by a system Reset.

Bits 7:3 Reserved, always read as 0.

Bit 2 **PVDO:** PVD output

This bit is set and cleared by hardware. It is valid only if PVD is enabled by the PVDE bit.

0: V_{DD}/V_{DDA} is higher than the PVD threshold selected with the PLS[2:0] bits.

1: V_{DD}/V_{DDA} is lower than the PVD threshold selected with the PLS[2:0] bits.

Note: 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.

Bit 1 **SBF:** Standby flag

This bit is set by hardware and cleared only by a POR/PDR (power on reset/power down reset) or by setting the CSBF bit in the *Power control register (PWR_CR)*

0: Device has not been in Standby mode

1: Device has been in Standby mode

Bit 0 **WUF:** Wakeup flag

This bit is set by hardware and cleared only by a POR/PDR (power on reset/power down reset) or by setting the CWUF bit in the *Power control register (PWR_CR)*

0: No wakeup event occurred

1: A wakeup event was received from the WKUP pin or from the RTC alarm

Note: 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.

4.4.3 PWR register map

The following table summarizes the PWR registers.

Table 12. PWR - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	**PWR_CR**	Reserved																							DBP	PLS[2:0]			PVDE	CSBF	CWUF	PDDS	LPDS
	Reset value																								0	0	0	0	0	0	0	0	0
0x004	**PWR_CSR**	Reserved																							EWUP	Reserved					PVDO	SBF	WUF
	Reset value																								0						0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

5 Backup registers (BKP)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

5.1 BKP introduction

The backup registers are forty two 16-bit registers for storing 84 bytes of user application data. They are implemented in the backup domain that remains powered on by V_{BAT} when the V_{DD} power is switched off. They are not reset when the device wakes up from Standby mode or by a system reset or power reset.

In addition, the BKP control registers are used to manage the Tamper detection feature and RTC calibration.

After reset, access to the Backup registers and RTC is disabled and the Backup domain (BKP) is protected against possible parasitic write access. To enable access to the Backup registers and the RTC, proceed as follows:

- enable the power and backup interface clocks by setting the PWREN and BKPEN bits in the RCC_APB1ENR register
- set the DBP bit the Power Control Register (PWR_CR) to enable access to the Backup registers and RTC.

5.2 BKP main features

- 20-byte data registers (in medium-density and low-density devices) or 84-byte data registers (in high-density devices)
- Status/control register for managing tamper detection with interrupt capability
- Calibration register for storing the RTC calibration value
- Possibility to output the RTC Calibration Clock, RTC Alarm pulse or Second pulse on TAMPER pin PC13 (when this pin is not used for tamper detection)

5.3 BKP functional description

5.3.1 Tamper detection

The TAMPER pin generates a Tamper detection event when the pin changes from 0 to 1 or from 1 to 0 depending on the TPAL bit in the *Backup control register (BKP_CR)*. A tamper detection event resets all data backup registers.

However to avoid losing Tamper events, the signal used for edge detection is logically ANDed with the Tamper enable in order to detect a Tamper event in case it occurs before the TAMPER pin is enabled.

- **When TPAL=0:** If the TAMPER pin is already high before it is enabled (by setting TPE bit), an extra Tamper event is detected as soon as the TAMPER pin is enabled (while there was no rising edge on the TAMPER pin after TPE was set)
- **When TPAL=1:** If the TAMPER pin is already low before it is enabled (by setting the TPE bit), an extra Tamper event is detected as soon as the TAMPER pin is enabled (while there was no falling edge on the TAMPER pin after TPE was set)

By setting the TPIE bit in the BKP_CSR register, an interrupt is generated when a Tamper detection event occurs.

After a Tamper event has been detected and cleared, the TAMPER pin should be disabled and then re-enabled with TPE before writing to the backup data registers (BKP_DRx) again. This prevents software from writing to the backup data registers (BKP_DRx), while the TAMPER pin value still indicates a Tamper detection. This is equivalent to a level detection on the TAMPER pin.

Note: Tamper detection is still active when V_{DD} power is switched off. To avoid unwanted resetting of the data backup registers, the TAMPER pin should be externally tied to the correct level.

5.3.2 RTC calibration

For measurement purposes, the RTC clock with a frequency divided by 64 can be output on the TAMPER pin. This is enabled by setting the CCO bit in the *RTC clock calibration register (BKP_RTCCR)*.

The clock can be slowed down by up to 121 ppm by configuring CAL[6:0] bits.

For more details about RTC calibration and how to use it to improve timekeeping accuracy, please refer to AN2604 "*STM32F101xx and STM32F103xx RTC calibration*".

5.4 BKP registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

5.4.1 Backup data register x (BKP_DRx) (x = 1 ..42)

Address offset: 0x04 to 0x28, 0x40 to 0xBC

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **D[15:0]** Backup data.

These bits can be written with user data.

Note: The BKP_DRx registers are not reset by a System reset or Power reset or when the device wakes up from Standby mode.
They are reset by a Backup Domain reset or by a TAMPER pin event (if the TAMPER pin function is activated).

5.4.2 RTC clock calibration register (BKP_RTCCR)

Address offset: 0x2C

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						ASOS	ASOE	CCO	CAL[6:0]						
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved, always read as 0.

Bit 9 **ASOS** Alarm or Second Output Selection

When the ASOE bit is set, the ASOS bit can be used to select whether the signal output on the TAMPER pin is the RTC Second pulse signal or the Alarm pulse signal:

0: RTC Alarm pulse output selected

1: RTC Second pulse output selected

Note: This bit is reset only by a Backup domain reset.

Bit 8 **ASOE** Alarm or second output enable

Setting this bit outputs either the RTC Alarm pulse signal or the Second pulse signal on the TAMPER pin depending on the ASOS bit.
The output pulse duration is one RTC clock period. The TAMPER pin must not be enabled while the ASOE bit is set.

Note: This bit is reset only by a Backup domain reset.

Bit 7 **CCO** Calibration clock output

0: No effect
1: Setting this bit outputs the RTC clock with a frequency divided by 64 on the TAMPER pin. The TAMPER pin must not be enabled while the CCO bit is set in order to avoid unwanted Tamper detection.

Note: This bit is reset when the V_{DD} supply is powered off.

Bit 6:0 **CAL[6:0]** *Calibration value*

This value indicates the number of clock pulses that will be ignored every 2^20 clock pulses. This allows the calibration of the RTC, slowing down the clock by steps of 1000000/2^20 PPM.
The clock of the RTC can be slowed down from 0 to 121PPM.

5.4.3 Backup control register (BKP_CR)

Address offset: 0x30

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														TPAL	TPE
														rw	rw

Bits 15:2 Reserved, always read as 0.

Bit 1 **TPAL** *TAMPER pin active level*

0: A high level on the TAMPER pin resets all data backup registers (if TPE bit is set).
1: A low level on the TAMPER pin resets all data backup registers (if TPE bit is set).

Bit 0 **TPE** *TAMPER pin enable*

0: The TAMPER pin is free for general purpose I/O
1: Tamper alternate I/O function is activated.

Note: Setting the TPAL and TPE bits at the same time is always safe, however resetting both at the same time can generate a spurious Tamper event. For this reason it is recommended to change the TPAL bit only when the TPE bit is reset.

5.4.4 Backup control/status register (BKP_CSR)

Address offset: 0x34

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						TIF	TEF	Reserved					TPIE	CTI	CTE
						r	r						rw	w	w

Bits 15:10 Reserved, always read as 0.

Bit 9 **TIF:** Tamper interrupt flag

This bit is set by hardware when a Tamper event is detected and the TPIE bit is set. It is cleared by writing 1 to the CTI bit (also clears the interrupt). It is also cleared if the TPIE bit is reset.

0: No Tamper interrupt

1: A Tamper interrupt occurred

Note: This bit is reset only by a system reset and wakeup from Standby mode.

Bit 8 **TEF:** Tamper event flag

This bit is set by hardware when a Tamper event is detected. It is cleared by writing 1 to the CTE bit.

0: No Tamper event

1: A Tamper event occurred

Note: A Tamper event resets all the BKP_DRx registers. They are held in reset as long as the TEF bit is set. If a write to the BKP_DRx registers is performed while this bit is set, the value will not be stored.

Bits 7:3 Reserved, always read as 0.

Bit 2 **TPIE:** TAMPER Pin interrupt enable

0: Tamper interrupt disabled

1: Tamper interrupt enabled (the TPE bit must also be set in the BKP_CR register

Note 1: A Tamper interrupt does not wake up the core from low-power modes.

Note 2: This bit is reset only by a system reset and wakeup from Standby mode.

Bit 1 **CTI:** Clear tamper interrupt

This bit is write only, and is always read as 0.

0: No effect

1: Clear the Tamper interrupt and the TIF Tamper interrupt flag.

Bit 0 **CTE:** Clear tamper event

This bit is write only, and is always read as 0.

0: No effect

1: Reset the TEF Tamper event flag (and the Tamper detector)

5.4.5 BKP register map

BKP registers are mapped as 16-bit addressable registers as described in the table below:

Table 13. BKP register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	Reserved																																
0x04	**BKP_DR1**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**BKP_DR2**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**BKP_DR3**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**BKP_DR4**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**BKP_DR5**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**BKP_DR6**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**BKP_DR7**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	**BKP_DR8**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	**BKP_DR9**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	**BKP_DR10**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2	**BKP_RTCCR**	Reserved																						ASOS	ASOE	CCO	CAL[6:0]						
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x30	**BKP_CR**	Reserved																														TPAL	TPE
	Reset value																															0	0
0x34	**BKP_CSR**	Reserved																						TIF	TEF	Reserved					TPIE	CTI	CTE
	Reset value																							0	0						0	0	0
0x38	Reserved																																
0x3C	Reserved																																
0x40	**BKP_DR11**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 13. BKP register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x44	**BKP_DR12**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x48	**BKP_DR13**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x4C	**BKP_DR14**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x50	**BKP_DR15**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x54	**BKP_DR16**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x58	**BKP_DR17**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x5C	**BKP_DR18**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x60	**BKP_DR19**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x64	**BKP_DR20**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x68	**BKP_DR21**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x6C	**BKP_DR22**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x70	**BKP_DR23**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x74	**BKP_DR24**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x78	**BKP_DR25**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x7C	**BKP_DR26**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x80	**BKP_DR27**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x84	**BKP_DR28**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 13. BKP register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x88	**BKP_DR29**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x8C	**BKP_DR30**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x90	**BKP_DR31**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x94	**BKP_DR32**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x98	**BKP_DR33**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x9C	**BKP_DR34**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xA0	**BKP_DR35**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xA4	**BKP_DR36**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xA8	**BKP_DR37**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xAC	**BKP_DR38**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xB0	**BKP_DR39**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xB4	**BKP_DR40**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xB8	**BKP_DR41**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0xBC	**BKP_DR42**	Reserved																D[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

6 Reset and clock control (RCC)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

6.1 Reset

There are three types of reset, defined as system Reset, power Reset and backup domain Reset.

6.1.1 System reset

A system reset sets all registers to their reset values except the reset flags in the clock controller CSR register and the registers in the Backup domain (see *Figure 3*).

A system reset is generated when one of the following events occurs:

1. A low level on the NRST pin (external reset)
2. Window watchdog end of count condition (WWDG reset)
3. Independent watchdog end of count condition (IWDG reset)
4. A software reset (SW reset) (see *Section : Software reset*)
5. Low-power management reset (see *Section : Low-power management reset*)

The reset source can be identified by checking the reset flags in the Control/Status register, RCC_CSR (see *Section 6.3.10: Control/status register (RCC_CSR)*).

Software reset

The SYSRESETREQ bit in Cortex™-M3 Application Interrupt and Reset Control Register must be set to force a software reset on the device. Refer to the Cortex™-M3 technical reference manual for more details.

Low-power management reset

There are two ways to generate a low-power management reset:

1. Reset generated when entering Standby mode:

   This type of reset is enabled by resetting nRST_STDBY bit in User Option Bytes. In this case, whenever a Standby mode entry sequence is successfully executed, the device is reset instead of entering Standby mode.
2. Reset when entering Stop mode:

   This type of reset is enabled by resetting NRST_STOP bit in User Option Bytes. In this case, whenever a Stop mode entry sequence is successfully executed, the device is reset instead of entering Stop mode.

For further information on the User Option Bytes, refer to the STM32F10xxx Flash programming manual.

6.1.2 Power reset

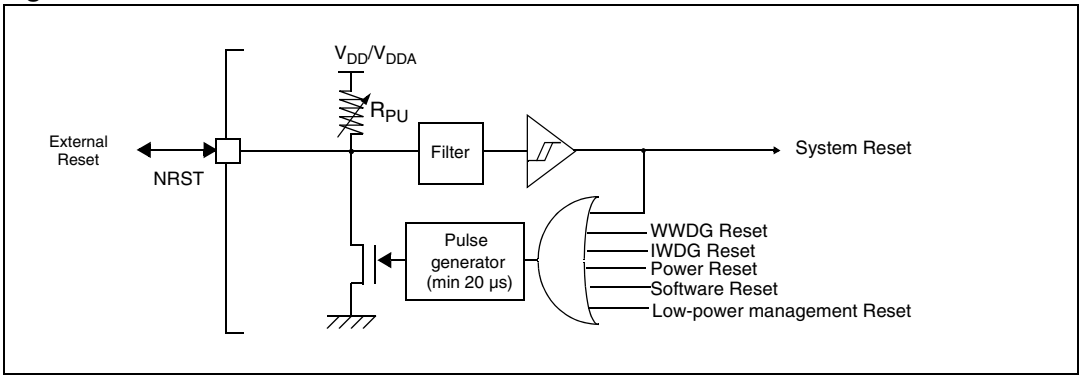
A power reset is generated when one of the following events occurs:

1. Power-on/power-down reset (POR/PDR reset)
2. When exiting Standby mode

A power reset sets all registers to their reset values except the Backup domain (see *Figure 3*)

These sources act on the NRST pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at address 0x0000_0004 in the memory map. For more details, refer to *Table 36: Vector table for other STM32F10xxx devices on page 117*.

Figure 6. Reset circuit



The Backup domain has two specific resets that affect only the Backup domain (see *Figure 3*).

6.1.3 Backup domain reset

A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the *Backup domain control register (RCC_BDCR)*.
2. V_{DD} or V_{BAT} power on, if both supplies have previously been powered off.

6.2 Clocks

Three different clock sources can be used to drive the system clock (SYSCLK):

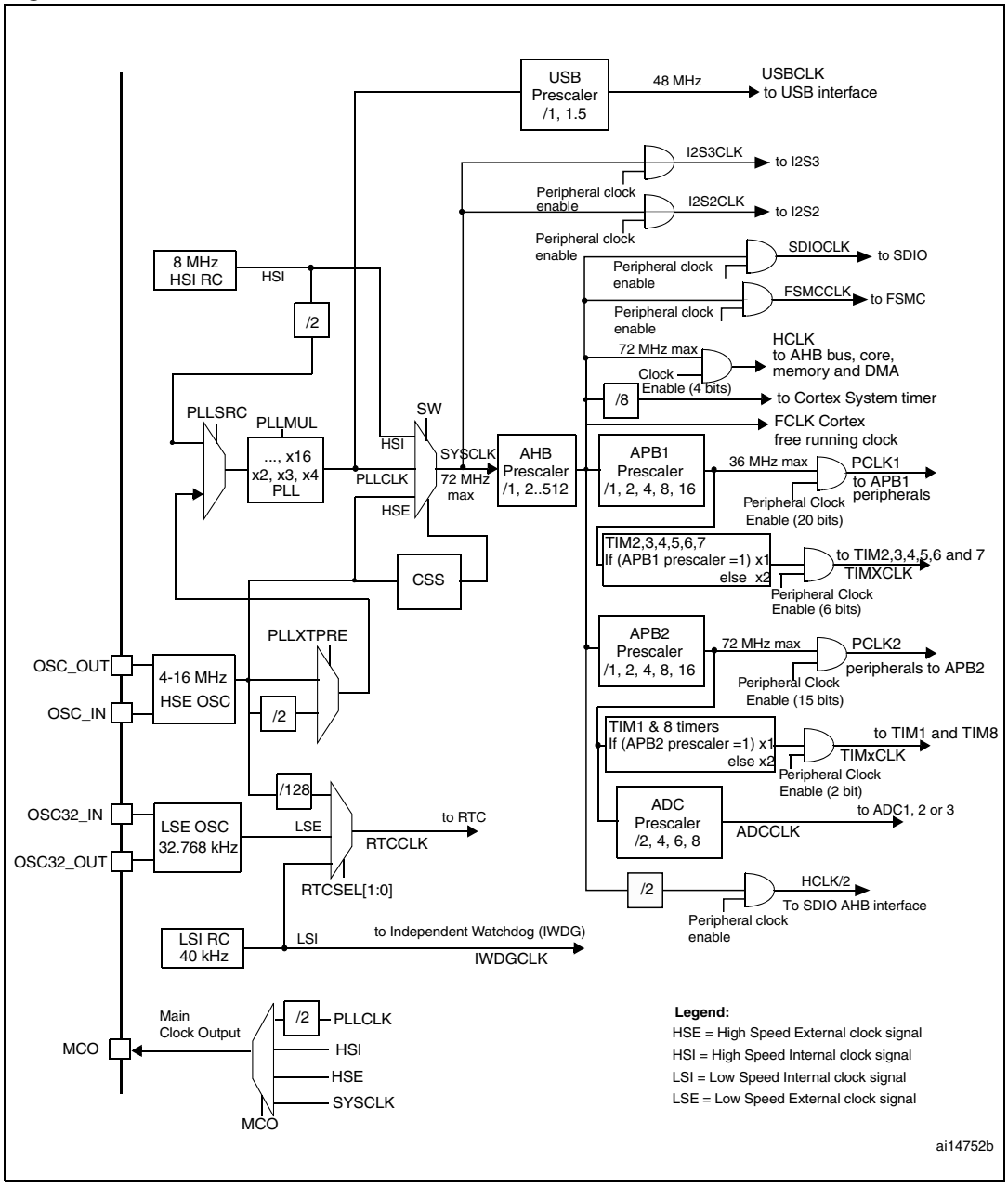
- HSI oscillator clock
- HSE oscillator clock
- PLL clock

The devices have the following two secondary clock sources:

- 40 kHz low speed internal RC (LSI RC) which drives the independent watchdog and optionally the RTC used for Auto-wakeup from Stop/Standby mode.
- 32.768 kHz low speed external crystal (LSE crystal) which optionally drives the real-time clock (RTCCLK)

Each clock source can be switched on or off independently when it is not used, to optimize power consumption.

Figure 7. Clock tree



1. When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.

Several prescalers allow the configuration of the AHB frequency, the high speed APB (APB2) and the low speed APB (APB1) domains. The maximum frequency of the AHB and the APB2 domains is 72 MHz. The maximum allowed frequency of the APB1 domain is 36 MHz. The SDIO AHB interface is clocked with a fixed frequency equal to HCLK/2.

The RCC feeds the Cortex System Timer (SysTick) external clock with the AHB clock divided by 8. The SysTick can work either with this clock or with the Cortex clock (AHB),

configurable in the SysTick Control and Status Register. The ADCs are clocked by the clock of the High Speed domain (APB2) divided by 2, 4, 6 or 8.

The timer clock frequencies are automatically fixed by hardware. There are two cases:

1. if the APB prescaler is 1, the timer clock frequencies are set to the same frequency as that of the APB domain to which the timers are connected.
2. otherwise, they are set to twice (×2) the frequency of the APB domain to which the timers are connected.

FCLK acts as Cortex™-M3 free running clock. For more details refer to the ARM Cortex™-M3 Technical Reference Manual.

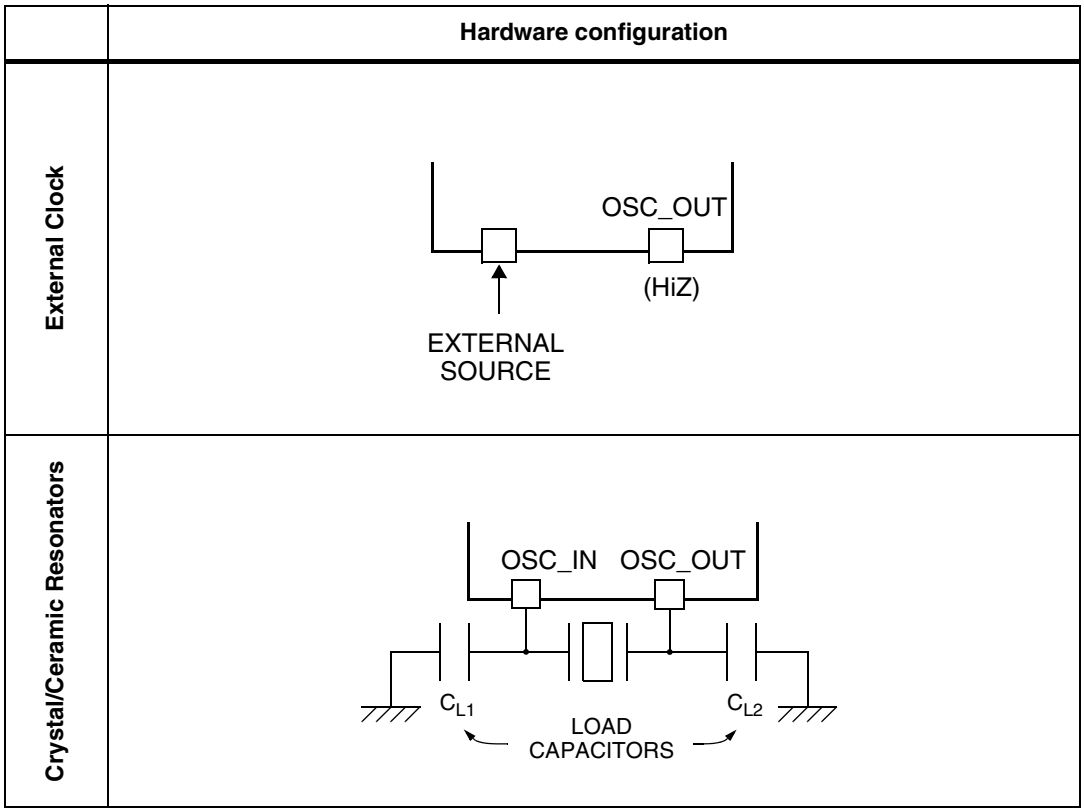
6.2.1 HSE clock

The high speed external clock signal (HSE) can be generated from two possible clock sources:

- HSE external crystal/ceramic resonator
- HSE user external clock

The resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and startup stabilization time. The loading capacitance values must be adjusted according to the selected oscillator.

Figure 8. HSE/ LSE clock sources



External source (HSE bypass)

In this mode, an external clock source must be provided. It can have a frequency of up to 25 MHz. You select this mode by setting the HSEBYP and HSEON bits in the *Clock control register (RCC_CR)*. The external clock signal (square, sinus or triangle) with ~50% duty cycle has to drive the OSC_IN pin while the OSC_OUT pin should be left hi-Z. See *Figure 8*.

External crystal/ceramic resonator (HSE crystal)

The 4 to 16 MHz external oscillator has the advantage of producing a very accurate rate on the main clock.

The associated hardware configuration is shown in *Figure 8*. Refer to the electrical characteristics section of the *datasheet* for more details.

The HSERDY flag in the *Clock control register (RCC_CR)* indicates if the high-speed external oscillator is stable or not. At startup, the clock is not released until this bit is set by hardware. An interrupt can be generated if enabled in the *Clock interrupt register (RCC_CIR)*.

The HSE Crystal can be switched on and off using the HSEON bit in the *Clock control register (RCC_CR)*.

6.2.2 HSI clock

The HSI clock signal is generated from an internal 8 MHz RC Oscillator and can be used directly as a system clock or divided by 2 to be used as PLL input.

The HSI RC oscillator has the advantage of providing a clock source at low cost (no external components). It also has a faster startup time than the HSE crystal oscillator however, even with calibration the frequency is less accurate than an external crystal oscillator or ceramic resonator.

Calibration

RC oscillator frequencies can vary from one chip to another due to manufacturing process variations, this is why each device is factory calibrated by ST for 1% accuracy at T_A=25°C.

After reset, the factory calibration value is loaded in the HSICAL[7:0] bits in the *Clock control register (RCC_CR)*.

If the application is subject to voltage or temperature variations this may affect the RC oscillator speed. You can trim the HSI frequency in the application using the HSITRIM[4:0] bits in the *Clock control register (RCC_CR)*.

The HSIRDY flag in the *Clock control register (RCC_CR)* indicates if the HSI RC is stable or not. At startup, the HSI RC output clock is not released until this bit is set by hardware.

The HSI RC can be switched on and off using the HSION bit in the *Clock control register (RCC_CR)*.

The HSI signal can also be used as a backup source (Auxiliary clock) if the HSE crystal oscillator fails. Refer to *Section 6.2.7: Clock security system (CSS) on page 72*.

6.2.3 PLL

The internal PLL can be used to multiply the HSI RC output or HSE crystal output clock frequency. Refer to *Figure 7* and *Clock control register (RCC_CR)*.

The PLL configuration (selection of HSI oscillator divided by 2 or HSE oscillator for PLL input clock, and multiplication factor) must be done before enabling the PLL. Once the PLL enabled, these parameters cannot be changed.

An interrupt can be generated when the PLL is ready if enabled in the *Clock interrupt register (RCC_CIR)*.

If the USB interface is used in the application, the PLL must be programmed to output 48 or 72 MHz. This is needed to provide a 48 MHz USBCLK.

6.2.4 LSE clock

The LSE crystal is a 32.768 kHz Low Speed External crystal or ceramic resonator. It has the advantage providing a low-power but highly accurate clock source to the real-time clock peripheral (RTC) for clock/calendar or other timing functions.

The LSE crystal is switched on and off using the LSEON bit in *Backup domain control register (RCC_BDCR)*.

The LSERDY flag in the *Backup domain control register (RCC_BDCR)* indicates if the LSE crystal is stable or not. At startup, the LSE crystal output clock signal is not released until this bit is set by hardware. An interrupt can be generated if enabled in the *Clock interrupt register (RCC_CIR)*.

External source (LSE bypass)

In this mode, an external clock source must be provided. It must have a frequency of 32.768 kHz. You select this mode by setting the LSEBYP and LSEON bits in the *Backup domain control register (RCC_BDCR)*. The external clock signal (square, sinus or triangle) with ~50% duty cycle has to drive the OSC32_IN pin while the OSC32_OUT pin should be left Hi-Z. See *Figure 8*.

6.2.5 LSI clock

The LSI RC acts as an low-power clock source that can be kept running in Stop and Standby mode for the independent watchdog (IWDG) and Auto-wakeup unit (AWU). The clock frequency is around 40 kHz (between 30 kHz and 60 kHz). For more details, refer to the electrical characteristics section of the datasheets.

The LSI RC can be switched on and off using the LSION bit in the *Control/status register (RCC_CSR)*.

The LSIRDY flag in the *Control/status register (RCC_CSR)* indicates if the low-speed internal oscillator is stable or not. At startup, the clock is not released until this bit is set by hardware. An interrupt can be generated if enabled in the *Clock interrupt register (RCC_CIR)*.

Note: LSI calibration is only available on high-density devices.

LSI calibration

The frequency dispersion of the Low Speed Internal RC (LSI) oscillator can be calibrated to have accurate RTC time base and/or IWDG timeout (when LSI is used as clock source for these peripherals) with an acceptable accuracy.

This calibration is performed by measuring the LSI clock frequency with respect to TIM5 input clock (TIM5CLK). According to this measurement done at the precision of the HSE

oscillator, the software can adjust the programmable 20-bit prescaler of the RTC to get an accurate time base or can compute accurate IWDG timeout.

Use the following procedure to calibrate the LSI:

1. Enable TIM5 timer and configure channel4 in input capture mode
2. Set the TIM5CH4_IREMAP bit in the AFIO_MAPR register to connect the LSI clock internally to TIM5 channel4 input capture for calibration purpose.
3. Measure the frequency of LSI clock using the TIM5 Capture/compare 4 event or interrupt.
4. Use the measured LSI frequency to update the 20-bit prescaler of the RTC depending on the desired time base and/or to compute the IWDG timeout.

6.2.6 System clock (SYSCLK) selection

After a system reset, the HSI oscillator is selected as system clock. When a clock source is used directly or through the PLL as system clock, it is not possible to stop it.

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. Status bits in the *Clock control register (RCC_CR)* indicate which clock(s) is (are) ready and which clock is currently used as system clock.

6.2.7 Clock security system (CSS)

Clock Security System can be activated by software. In this case, the clock detector is enabled after the HSE oscillator startup delay, and disabled when this oscillator is stopped.

If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled, a clock failure event is sent to the break input of the TIM1 Advanced control timer and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex™-M3 NMI (Non-Maskable Interrupt) exception vector.

Note: 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 unless the CSS interrupt pending bit is cleared. As a consequence, in the NMI ISR user must clear the CSS interrupt by setting the CSSC bit in the Clock interrupt register (RCC_CIR).

If the HSE oscillator is used directly or indirectly as the system clock (indirectly means: it is used as PLL input clock, and the PLL clock is used as system clock), a detected failure causes a switch of the system clock to the HSI oscillator and the disabling of the external HSE oscillator. If the HSE oscillator clock (divided or not) is the clock entry of the PLL used as system clock when the failure occurs, the PLL is disabled too.

6.2.8 RTC clock

The RTCCLK clock source can be either the HSE/128, LSE or LSI clocks. This is selected by programming the RTCSEL[1:0] bits in the *Backup domain control register (RCC_BDCR)*. This selection cannot be modified without resetting the Backup domain.

The LSE clock is in the Backup domain, whereas the HSE and LSI clocks are not. Consequently:

- If LSE is selected as RTC clock:
  - The RTC continues to work even if the V_{DD} supply is switched off, provided the V_{BAT} supply is maintained.
- If LSI is selected as Auto-Wakeup unit (AWU) clock:
  - The AWU state is not guaranteed if the V_{DD} supply is powered off. Refer to *Section 6.2.5: LSI clock on page 71* for more details on LSI calibration.
- If the HSE clock divided by 128 is used as RTC clock:
  - The RTC state is not guaranteed if the V_{DD} supply is powered off or if the internal voltage regulator is powered off (removing power from the 1.8 V domain).

6.2.9 Watchdog clock

If the Independent watchdog (IWDG) is started by either hardware option or software access, the LSI oscillator is forced ON and cannot be disabled. After the LSI oscillator temporization, the clock is provided to the IWDG.

6.2.10 Clock-out capability

The microcontroller clock output (MCO) capability allows the clock to be output onto the external MCO pin. The configuration registers of the corresponding GPIO port must be programmed in alternate function mode. One of 4 clock signals can be selected as the MCO clock.

- SYSCLK
- HSI
- HSE
- PLL clock divided by 2

The selection is controlled by the MCO[2:0] bits of the *Clock configuration register (RCC_CFGR)*.

6.3 RCC registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

6.3.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bits 31:26 Reserved, always read as 0.

Bit 25 **PLLRDY:** PLL clock ready flag

Set by hardware to indicate that the PLL is locked.
0: PLL unlocked
1: PLL locked

Bit 24 **PLLON:** PLL enable

Set and cleared by software to enable PLL.
Cleared by hardware when entering Stop or Standby mode. This bit can not be reset if the PLL clock is used as system clock or is selected to become the system clock.
0: PLL OFF
1: PLL ON

Bits 23:20 Reserved, always read as 0.

Bit 19 **CSSON:** Clock security system enable

Set and cleared by software to enable clock detector.
0: Clock detector OFF
1: Clock detector ON if external 4-25 MHz oscillator is ready.

Bit 18 **HSEBYP:** External high-speed clock bypass

Set and cleared by software in debug for bypassing the oscillator with an external clock. This bit can be written only if the external 4-25 MHz oscillator is disabled.
0: external 4-25 MHz oscillator not bypassed
1: external 4-25 MHz oscillator bypassed with external clock

Bit 17 **HSERDY:** External high-speed clock ready flag

Set by hardware to indicate that the external 4-25 MHz oscillator is stable. This bit needs 6 cycles of external 4-25 MHz oscillator clock to fall down after HSEON reset.
0: external 4-25 MHz oscillator not ready
1: external 4-25 MHz oscillator ready

Bit 16 **HSEON:** External high-speed clock enable

Set and cleared by software.

Cleared by hardware to stop the external 1-25MHz oscillator when entering in Stop or Standby mode. This bit cannot be reset if the external 4-25 MHz oscillator is used directly or indirectly as the system clock or is selected to become the system clock.

0: HSE oscillator OFF

1: HSE oscillator ON

Bits 15:8 **HSICAL[7:0]:** Internal high-speed clock calibration

These bits are initialized automatically at startup.

Bits 7:3 **HSITRIM[4:0]:** Internal high-speed clock trimming

These bits provide an additional user-programmable trimming value that is added to the HSICAL[7:0] bits. It can be programmed to adjust to variations in voltage and temperature that influence the frequency of the internal HSI RC.

The default value is 16, which, when added to the HSICAL value, should trim the HSI to 8 MHz ± 1%. The trimming step ($F_{hsitrim}$) is around 40 kHz between two consecutive HSICAL steps.

Bit 2 Reserved, always read as 0.

Bit 1 **HSIRDY:** Internal high-speed clock ready flag

Set by hardware to indicate that internal 8 MHz RC oscillator is stable. After the HSION bit is cleared, HSIRDY goes low after 6 internal 8 MHz RC oscillator clock cycles.

0: internal 8 MHz RC oscillator not ready

1: internal 8 MHz RC oscillator ready

Bit 0 **HSION:** Internal high-speed clock enable

Set and cleared by software.

Set by hardware to force the internal 8 MHz RC oscillator ON when leaving Stop or Standby mode or in case of failure of the external 4-25 MHz oscillator used directly or indirectly as system clock. This bit cannot be reset if the internal 8 MHz RC is used directly or indirectly as system clock or is selected to become the system clock.

0: internal 8 MHz RC oscillator OFF

1: internal 8 MHz RC oscillator ON

6.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 31:27 Reserved, always read as 0.

Bits 26:24 **MCO:** Microcontroller clock output

Set and cleared by software.

0xx: No clock

100: System clock (SYSCLK) selected

101: HSI clock selected

110: HSE clock selected

111: PLL clock divided by 2 selected

Note: This clock output may have some truncated cycles at startup or during MCO clock source switching.

When the System Clock is selected to output to the MCO pin, make sure that this clock does not exceed 50 MHz (the maximum I/O speed).

Bit 22 **USBPRE:** USB prescaler

Set and cleared by software to generate 48 MHz USB clock. This bit must be valid before enabling the USB clock in the RCC_APB1ENR register. This bit can't be reset if the USB clock is enabled.

0: PLL clock is divided by 1.5

1: PLL clock is not divided

Bits 21:18 **PLLMUL:** PLL multiplication factor

These bits are written by software to define the PLL multiplication factor. These bits can be written only when PLL is disabled.

Caution: The PLL output frequency must not exceed 72 MHz.

0000: PLL input clock x 2

0001: PLL input clock x 3

0010: PLL input clock x 4

0011: PLL input clock x 5

0100: PLL input clock x 6

0101: PLL input clock x 7

0110: PLL input clock x 8

0111: PLL input clock x 9

1000: PLL input clock x 10

1001: PLL input clock x 11

1010: PLL input clock x 12

1011: PLL input clock x 13

1100: PLL input clock x 14

1101: PLL input clock x 15

1110: PLL input clock x 16

1111: PLL input clock x 16

Bit 17 **PLLXTPRE:** HSE divider for PLL entry

Set and cleared by software to divide HSE before PLL entry. This bit can be written only when PLL is disabled.

0: HSE clock not divided

1: HSE clock divided by 2

Bit 16 **PLLSRC:** PLL entry clock source

Set and cleared by software to select PLL clock source. This bit can be written only when PLL is disabled.

0: HSI oscillator clock / 2 selected as PLL input clock

1: HSE oscillator clock selected as PLL input clock

Bits 14:14 **ADCPRE:** ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADCs.
00: PLCK2 divided by 2
01: PLCK2 divided by 4
10: PLCK2 divided by 6
11: PLCK2 divided by 8

Bits 13:11 **PPRE2:** APB high-speed prescaler (APB2)

Set and cleared by software to control the division factor of the APB high-speed clock (PCLK2).
0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Bits 10:8 **PPRE1:** APB low-speed prescaler (APB1)

Set and cleared by software to control the division factor of the APB low-speed clock (PCLK1).
Warning: the software has to set correctly these bits to not exceed 36 MHz on this domain.
0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Bits 7:4 **HPRE:** AHB prescaler

Set and cleared by software to control the division factor of the AHB clock.
0xxx: SYSCLK not divided
1000: SYSCLK divided by 2
1001: SYSCLK divided by 4
1010: SYSCLK divided by 8
1011: SYSCLK divided by 16
1100: SYSCLK divided by 64
1101: SYSCLK divided by 128
1110: SYSCLK divided by 256
1111: SYSCLK divided by 512
Note: The prefetch buffer must be kept on when using a prescaler different from 1 on the AHB clock. Refer to *Reading Flash memory on page 40* section for more details.

Bits 3:2 **SWS:** System clock switch status

Set and cleared by hardware to indicate which clock source is used as system clock.
00: HSI oscillator used as system clock
01: HSE oscillator used as system clock
10: PLL used as system clock
11: not applicable

Bits 1:0 **SW:** System clock switch

Set and cleared by software to select SYSCLK source.

Set by hardware to force HSI selection when leaving Stop and Standby mode or in case of failure of the HSE oscillator used directly or indirectly as system clock (if the Clock Security System is enabled).

00: HSI selected as system clock

01: HSE selected as system clock

10: PLL selected as system clock

11: not allowed

6.3.3 Clock interrupt register (RCC_CIR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								CSSC	Reserved		PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC
								w			w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	Reserved		PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF
			rw	rw	rw	rw	rw	r			r	r	r	r	r

Bits 31:24 Reserved, always read as 0.

Bit 23 **CSSC:** Clock security system interrupt clear

This bit is set by software to clear the CSSF flag.

0: No effect

1: Clear CSSF flag

Bits 22:21 Reserved, always read as 0.

Bit 20 **PLLRDYC:** PLL ready interrupt clear

This bit is set by software to clear the PLLRDYF flag.

0: No effect

1: PLLRDYF cleared

Bit 19 **HSERDYC:** HSE ready interrupt clear

This bit is set by software to clear the HSERDYF flag.

0: No effect

1: HSERDYF cleared

Bit 18 **HSIRDYC:** HSI ready interrupt clear

This bit is set software to clear the HSIRDYF flag.

0: No effect

1: HSIRDYF cleared

Bit 17 **LSERDYC:** LSE ready interrupt clear

This bit is set by software to clear the LSERDYF flag.

0: No effect

1: LSERDYF cleared

Bit 16 **LSIRDYC:** LSI ready interrupt clear
This bit is set by software to clear the LSIRDYF flag.
0: No effect
1: LSIRDYF cleared

Bits 15:13 Reserved, always read as 0.

Bit 12 **PLLRDYIE:** PLL ready interrupt enable
Set and cleared by software to enable/disable interrupt caused by PLL lock.
0: PLL lock interrupt disabled
1: PLL lock interrupt enabled

Bit 11 **HSERDYIE:** HSE ready interrupt enable
Set and cleared by software to enable/disable interrupt caused by the external 4-25 MHz oscillator stabilization.
0: HSE ready interrupt disabled
1: HSE ready interrupt enabled

Bit 10 **HSIRDYIE:** HSI ready interrupt enable
Set and cleared by software to enable/disable interrupt caused by the internal 8 MHz RC oscillator stabilization.
0: HSI ready interrupt disabled
1: HSI ready interrupt enabled

Bit 9 **LSERDYIE:** LSE ready interrupt enable
Set and cleared by software to enable/disable interrupt caused by the external 32 kHz oscillator stabilization.
0: LSE ready interrupt disabled
1: LSE ready interrupt enabled

Bit 8 **LSIRDYIE:** LSI ready interrupt enable
Set and cleared by software to enable/disable interrupt caused by internal RC 40 kHz oscillator stabilization.
0: LSI ready interrupt disabled
1: LSI ready interrupt enabled

Bit 7 **CSSF:** Clock security system interrupt flag
Set by hardware when a failure is detected in the external 4-25 MHz oscillator.
Cleared by software setting the CSSC bit.
0: No clock security interrupt caused by HSE clock failure
1: Clock security interrupt caused by HSE clock failure

Bits 6:5 Reserved, always read as 0.

Bit 4 **PLLRDYF:** PLL ready interrupt flag
Set by hardware when the PLL locks and PLLRDYDIE is set.
Cleared by software setting the PLLRDYC bit.
0: No clock ready interrupt caused by PLL lock
1: Clock ready interrupt caused by PLL lock

Bit3 **HSERDYF:** HSE ready interrupt flag
Set by hardware when External Low Speed clock becomes stable and HSERDYDIE is set.
Cleared by software setting the HSERDYC bit.
0: No clock ready interrupt caused by the external 4-25 MHz oscillator
1: Clock ready interrupt caused by the external 4-25 MHz oscillator

Bit 2 **HSIRDYF:** HSI ready interrupt flag

Set by hardware when the Internal High Speed clock becomes stable and HSIRDYDIE is set.

Cleared by software setting the HSIRDYC bit.

0: No clock ready interrupt caused by the internal 8 MHz RC oscillator

1: Clock ready interrupt caused by the internal 8 MHz RC oscillator

Bit 1 **LSERDYF:** LSE ready interrupt flag

Set by hardware when the External Low Speed clock becomes stable and LSERDYDIE is set.

Cleared by software setting the LSERDYC bit.

0: No clock ready interrupt caused by the external 32 kHz oscillator

1: Clock ready interrupt caused by the external 32 kHz oscillator

Bit 0 **LSIRDYF:** LSI ready interrupt flag

Set by hardware when the internal low speed clock becomes stable and LSIRDYDIE is set.

Cleared by software setting the LSIRDYC bit.

0: No clock ready interrupt caused by the internal RC 40 kHz oscillator

1: Clock ready interrupt caused by the internal RC 40 kHz oscillator

6.3.4 APB2 peripheral reset register (RCC_APB2RSTR)

Address offset: 0x0C

Reset value: 0x00000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 RST	USART1 RST	TIM8 RST	SPI1 RST	TIM1 RST	ADC2 RST	ADC1 RST	IOPG RST	IOPF RST	IOPE RST	IOPD RST	IOPC RST	IOPB RST	IOPA RST	Res.	AFIO RST
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.	rw

Bits 31:16 Reserved, always read as 0.

Bit 15 **ADC3RST:** ADC3 interface reset

Set and cleared by software.

0: No effect

1: Reset ADC3 interface

Bit 14 **USART1RST:** USART1 reset

Set and cleared by software.

0: No effect

1: Reset USART1

Bit 13 **TIM8RST:** TIM8 timer reset

Set and cleared by software.

0: No effect

1: Reset TIM8 timer

Bit 12 **SPI1RST:** SPI 1 reset
Set and cleared by software.
0: No effect
1: Reset SPI 1

Bit 11 **TIM1RST:** TIM1 timer reset
Set and cleared by software.
0: No effect
1: Reset TIM1 timer

Bit 10 **ADC2RST:** ADC 2 interface reset
Set and cleared by software.
0: No effect
1: Reset ADC 2 interface

Bit 9 **ADC1RST:** ADC 1 interface reset
Set and cleared by software.
0: No effect
1: Reset ADC 1 interface

Bit 8 **IOPGRST:** IO port G reset
Set and cleared by software.
0: No effect
1: Reset IO port G

Bit 7 **IOPFRST:** IO port F reset
Set and cleared by software.
0: No effect
1: Reset IO port F

Bit 6 **IOPERST:** IO port E reset
Set and cleared by software.
0: No effect
1: Reset IO port E

Bit 5 **IOPDRST:** IO port D reset
Set and cleared by software.
0: No effect
1: Reset I/O port D

Bit 4 **IOPCRST:** IO port C reset
Set and cleared by software.
0: No effect
1: Reset I/O port C

Bit 3 **IOPBRST:** IO port B reset
Set and cleared by software.
0: No effect
1: Reset I/O port B

Bit 2 **IOPARST:** I/O port A reset
Set and cleared by software.
0: No effect
1: Reset I/O port A

Bit 1 Reserved, always read as 0.

Bit 0 **AFIORST:** Alternate function I/O reset
Set and cleared by software.
0: No effect
1: Reset Alternate Function

6.3.5 APB1 peripheral reset register (RCC_APB1RSTR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC RST	PWR RST	BKP RST	Res.	CAN RST	Res.	USB RST	I2C2 RST	I2C1 RST	UART 5 RST	UART 4 RST	USART 3 RST	USART 2 RST	Res.
		rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 RST	SPI2 RST	Reserved		WWD GRST	Reserved					TIM7 RST	TIM6 RST	TIM5 RST	TIM4 RST	TIM3 RST	TIM2 RST
rw	rw			rw						rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, always read as 0.

Bit 29 **DACRST:** DAC interface reset
Set and cleared by software.
0: No effect
1: Reset DAC interface

Bit 28 **PWRRST:** Power interface reset
Set and cleared by software.
0: No effect
1: Reset power interface

Bit 27 **BKPRST:** Backup interface reset
Set and cleared by software.
0: No effect
1: Reset backup interface

Bit 26 Reserved, always read as 0.

Bit 25 **CANRST:** CAN reset
Set and cleared by software.
0: No effect
1: Reset CAN

Bit 24 Reserved, always read as 0.

Bit 23 **USBRST:** USB reset
Set and cleared by software.
0: No effect
1: Reset USB

Bit 22 **I2C2RST:** I2C 2 reset
Set and cleared by software.
0: No effect
1: Reset I2C 2

Bit 21 **I2C1RST:** I2C 1 reset
Set and cleared by software.
0: No effect
1: Reset I2C 1

Bit 20 **UART5RST:** USART 5 reset
Set and cleared by software.
0: No effect
1: Reset USART 5

Bit 19 **UART4RST:** USART 4 reset
Set and cleared by software.
0: No effect
1: Reset USART 4

Bit 18 **USART3RST:** USART 3 reset
Set and cleared by software.
0: No effect
1: Reset USART 3

Bit 17 **USART2RST:** USART 2 reset
Set and cleared by software.
0: No effect
1: Reset USART 2

Bits 16 Reserved, always read as 0.

Bit 15 **SPI3RST:** SPI 3 reset
Set and cleared by software.
0: No effect
1: Reset SPI 3

Bit 14 **SPI2RST:** SPI 2 reset
Set and cleared by software.
0: No effect
1: Reset SPI 2

Bits 13:12 Reserved, always read as 0.

Bit 11 **WWDGRST:** Window watchdog reset
Set and cleared by software.
0: No effect
1: Reset window watchdog

Bits 10:6 Reserved, always read as 0.

Bit 5 **TIM7RST:** Timer 7 reset
Set and cleared by software.
0: No effect
1: Reset timer 7

Bit 4 **TIM6RST:** Timer 6 reset
Set and cleared by software.
0: No effect
1: Reset timer 6

Bit 3 **TIM5RST:** Timer 5 reset
Set and cleared by software.
0: No effect
1: Reset timer 5

Bit 2 **TIM4RST:** Timer 4 reset
Set and cleared by software.
0: No effect
1: Reset timer 4

Bit 1 **TIM3RST:** Timer 3 reset
Set and cleared by software.
0: No effect
1: Reset timer 3

Bit 0 **TIM2RST:** Timer 2 reset
Set and cleared by software.
0: No effect
1: Reset timer 2

6.3.6 AHB peripheral clock enable register (RCC_AHBENR)

Address offset: 0x14

Reset value: 0x0000 0014

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					SDIO EN	Res.	FSMC EN	Res.	CRCE N	Res.	FLITF EN	Res.	SRAM EN	DMA2 EN	DMA1 EN
					rw		rw		rw		rw		rw	rw	rw

Bits 31:11 Reserved, always read as 0.

Bit 10 **SDIOEN:** SDIO clock enable
Set and cleared by software.
0: SDIO clock disabled
1: SDIO clock enabled

Bits 9 Reserved, always read as 0.

Bit 8 **FSMCEN:** FSMC clock enable
Set and cleared by software.
0: FSMC clock disabled
1: FSMC clock enabled

Bit 7 Reserved, always read as 0.

Bit 6 **CRCEN:** CRC clock enable
Set and cleared by software.
0: CRC clock disabled
1: CRC clock enabled

Bit 5 Reserved, always read as 0.

Bit 4 **FLITFEN:** FLITF clock enable
Set and cleared by software to disable/enable FLITF clock during sleep mode.
0: FLITF clock disabled during Sleep mode
1: FLITF clock enabled during Sleep mode

Bit 3 Reserved, always read as 0.

Bit 2 **SRAMEN:** SRAM interface clock enable
Set and cleared by software to disable/enable SRAM interface clock during Sleep mode.
0: SRAM interface clock disabled during Sleep mode.
1: SRAM interface clock enabled during Sleep mode

Bit 1 **DMA2EN:** DMA2 clock enable
Set and cleared by software.
0: DMA2 clock disabled
1: DMA2 clock enabled

Bit 0 **DMA1EN:** DMA1 clock enable
Set and cleared by software.
0: DMA1 clock disabled
1: DMA1 clock enabled

6.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bits 31:16 Reserved, always read as 0.

Bit 15 **ADC3EN:** ADC 3 interface clock enable
Set and cleared by software.
0: ADC 3 interface clock disabled
1: ADC 3 interface clock enabled

Bit 14 **USART1EN:** USART1 clock enable
Set and cleared by software.
0: USART1 clock disabled
1: USART1 clock enabled

Bit 13 **TIM8EN:** TIM8 Timer clock enable
Set and cleared by software.
0: TIM8 timer clock disabled
1: TIM8 timer clock enabled

Bit 12 **SPI1EN:** SPI 1 clock enable
Set and cleared by software.
0: SPI 1 clock disabled
1: SPI 1 clock enabled

Bit 11 **TIM1EN:** TIM1 Timer clock enable
Set and cleared by software.
0: TIM1 timer clock disabled
1: TIM1 timer clock enabled

Bit 10 **ADC2EN:** ADC 2 interface clock enable
Set and cleared by software.
0: ADC 2 interface clock disabled
1: ADC 2 interface clock enabled

Bit 9 **ADC1EN:** ADC 1 interface clock enable
Set and cleared by software.
0: ADC 1 interface disabled
1: ADC 1 interface clock enabled

Bit 8 **IOPGEN:** I/O port G clock enable
Set and cleared by software.
0: I/O port G clock disabled
1: I/O port G clock enabled

Bit 7 **IOPFEN:** I/O port F clock enable
Set and cleared by software.
0: I/O port F clock disabled
1: I/O port F clock enabled

Bit 6 **IOPEEN:** I/O port E clock enable
Set and cleared by software.
0: I/O port E clock disabled
1: I/O port E clock enabled

Bit 5 **IOPDEN:** I/O port D clock enable
Set and cleared by software.
0: I/O port D clock disabled
1: I/O port D clock enabled

Bit 4 **IOPCEN:** I/O port C clock enable
Set and cleared by software.
0: I/O port C clock disabled
1:I/O port C clock enabled

Bit 3 **IOPBEN:** I/O port B clock enable
Set and cleared by software.
0: I/O port B clock disabled
1:I/O port B clock enabled

Bit 2 **IOPAEN:** I/O port A clock enable
Set and cleared by software.
0: I/O port A clock disabled
1:I/O port A clock enabled

Bit 1 Reserved, always read as 0.

Bit 0 **AFIOEN:** Alternate function I/O clock enable
Set and cleared by software.
0: Alternate Function I/O clock disabled
1:Alternate Function I/O clock enabled

6.3.8 APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	UART5E N	UART4 EN	USART 3EN	USART 2EN	Res.
Res.		rw	rw	rw	Res.	rw	Res.	rw	rw	rw	rw	rw	rw	rw	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw	Res.		rw	Res.					rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, always read as 0.

Bit 29 **DACEN:** DAC interface clock enable
Set and cleared by software.
0: DAC interface clock disabled
1: DAC interface clock enable

Bit 28 **PWREN:** Power interface clock enable
Set and cleared by software.
0: Power interface clock disabled
1: Power interface clock enable

Bit 27 **BKPEN:** Backup interface clock enable
Set and cleared by software.
0: Backup interface clock disabled
1: Backup interface clock enabled

Bit 26 Reserved, always read as 0.

Bit 25 **CANEN:** CAN clock enable
Set and cleared by software.
0: CAN clock disabled
1: CAN clock enabled

Bit 24 Reserved, always read as 0.

Bit 23 **USBEN:** USB clock enable
Set and cleared by software.
0: USB clock disabled
1: USB clock enabled

Bit 22 **I2C2EN:** I2C 2 clock enable
Set and cleared by software.
0: I2C 2 clock disabled
1: I2C 2 clock enabled

Bit 21 **I2C1EN:** I2C 1 clock enable
Set and cleared by software.
0: I2C 1 clock disabled
1: I2C 1 clock enabled

Bit 20 **UART5EN:** USART 5 clock enable
Set and cleared by software.
0: USART 5 clock disabled
1: USART 5 clock enabled

Bit 19 **UART4EN:** USART 4 clock enable
Set and cleared by software.
0: USART 4 clock disabled
1: USART 4 clock enabled

Bit 18 **USART3EN:** USART 3 clock enable
Set and cleared by software.
0: USART 3 clock disabled
1: USART 3 clock enabled

Bit 17 **USART2EN:** USART 2 clock enable
Set and cleared by software.
0: USART 2 clock disabled
1: USART 2 clock enabled

Bits 16 Reserved, always read as 0.

Bit 15 **SPI3EN:** SPI 3 clock enable
Set and cleared by software.
0: SPI 3 clock disabled
1: SPI 3 clock enabled

Bit 14 **SPI2EN:** SPI 2 clock enable
Set and cleared by software.
0: SPI 2 clock disabled
1: SPI 2 clock enabled

Bits 13:12 Reserved, always read as 0.

Bit 11 **WWDGEN:** Window watchdog clock enable
Set and cleared by software.
0: Window watchdog clock disabled
1: Window watchdog clock enabled

Bits 10:6 Reserved, always read as 0.

Bit 5 **TIM7EN:** Timer 7 clock enable
Set and cleared by software.
0: Timer 7 clock disabled
1: Timer 7 clock enabled

Bit 4 **TIM6EN:** Timer 6 clock enable
Set and cleared by software.
0: Timer 6 clock disabled
1: Timer 6 clock enabled

Bit 3 **TIM5EN:** Timer 5 clock enable
Set and cleared by software.
0: Timer 5 clock disabled
1: Timer 5 clock enabled

Bit 2 **TIM4EN:** Timer 4 clock enable
Set and cleared by software.
0: Timer 4 clock disabled
1: Timer 4 clock enabled

Bit 1 **TIM3EN:** Timer 3 clock enable
Set and cleared by software.
0: Timer 3 clock disabled
1: Timer 3 clock enabled

Bit 0 **TIM2EN:** Timer 2 clock enable
Set and cleared by software.
0: Timer 2 clock disabled
1: Timer 2 clock enabled

6.3.9 Backup domain control register (RCC_BDCR)

Address offset: 0x20
Reset value: 0x0000 0000, reset by Backup domain Reset.
Access: 0 ≤ wait state ≤ 3, word, half-word and byte access
Wait states are inserted in case of successive accesses to this register.

Note: LSEON, LSEBYP, RTCSEL and RTCEN bits of the Backup domain control register (RCC_BDCR) are in the Backup domain. As a result, after Reset, these bits are write-protected and the DBP bit in the Power control register (PWR_CR) has to be set before these can be modified. Refer to Section 5 on page 58 for further information. These bits are only reset after a Backup domain Reset (see Section 6.1.3: Backup domain reset). Any internal or external Reset will not have any effect on these bits.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															BDRST
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC EN	Reserved					RTCSEL[1:0]		Reserved					LSE BYP	LSE RDY	LSEON
rw						rw	rw						rw	r	rw

Bits 31:17 Reserved, always read as 0.

Bit 16 **BDRST:** Backup domain software reset
Set and cleared by software.
0: Reset not activated
1: Resets the entire Backup domain

Bit 15 **RTCEN:** RTC clock enable
Set and cleared by software.
0: RTC clock disabled
1: RTC clock enabled

Bits 14:10 Reserved, always read as 0.

Bits 9:8 **RTCSEL[1:0]:** RTC clock source selection
Set by software to select the clock source for the RTC. Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset. The BDRST bit can be used to reset them.
00: No clock
01: LSE oscillator clock used as RTC clock
10: LSI oscillator clock used as RTC clock
11: HSE oscillator clock divided by 128 used as RTC clock

Bits 7:3 Reserved, always read as 0.

Bit 2 **LSEBYP:** External low-speed oscillator bypass
Set and cleared by software to bypass oscillator in debug mode. This bit can be written only when the external 32 kHz oscillator is disabled.
0: LSE oscillator not bypassed
1: LSE oscillator bypassed

Bit 1 **LSERDY:** External low-speed oscillator ready
Set and cleared by hardware to indicate when the external 32 kHz oscillator is stable. After the LSEON bit is cleared, LSERDY goes low after 6 external low-speed oscillator clock cycles.
0: External 32 kHz oscillator not ready
1: External 32 kHz oscillator ready

Bit 0 **LSEON:** External low-speed oscillator enable
Set and cleared by software.
0: External 32 kHz oscillator OFF
1: External 32 kHz oscillator ON

6.3.10 Control/status register (RCC_CSR)

Address: 0x24

Reset value: 0x0C00 0000, reset by system Reset, except reset flags by power Reset only.

Access: 0 ≤ wait state ≤ 3, word, half-word and byte access

Wait states are inserted in case of successive accesses to this register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPWR RSTF	WWDG RSTF	IWDG RSTF	SFT RSTF	POR RSTF	PIN RSTF	Res.	RMVF	Reserved							
rw	rw	rw	rw	rw	rw		rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														LSI RDY	LSION
														r	rw

Bit 31 **LPWRRSTF:** Low-power reset flag

Set by hardware when a Low-power management reset occurs.
Cleared by writing to the RMVF bit.
0: No Low-power management reset occurred
1: Low-power management reset occurred
For further information on Low-power management reset, refer to *Section : Low-power management reset*.

Bit 30 **WWDGRSTF:** Window watchdog reset flag

Set by hardware when a window watchdog reset occurs.
Cleared by writing to the RMVF bit.
0: No window watchdog reset occurred
1: Window watchdog reset occurred

Bit 29 **IWDGRSTF:** Independent watchdog reset flag

Set by hardware when an independent watchdog reset from V_{DD} domain occurs.
Cleared by writing to the RMVF bit.
0: No watchdog reset occurred
1: Watchdog reset occurred

Bit 28 **SFTRSTF:** Software reset flag

Set by hardware when a software reset occurs.
Cleared by writing to the RMVF bit.
0: No software reset occurred
1: Software reset occurred

Bit 27 **PORRSTF:** POR/PDR reset flag

Set by hardware when a POR/PDR reset occurs.
Cleared by writing to the RMVF bit.
0: No POR/PDR reset occurred
1: POR/PDR reset occurred

Bit 26 **PINRSTF:** PIN reset flag

Set by hardware when a reset from the NRST pin occurs.
Cleared by writing to the RMVF bit.
0: No reset from NRST pin occurred
1: Reset from NRST pin occurred

Bit 25 Reserved, always read as 0.

Bit 24 **RMVF:** Remove reset flag

Set by software to clear the reset flags.

0: No effect

1: Clear the reset flags

Bits 23:2 Reserved, always read as 0.

Bit 1 **LSIRDY:** Internal low-speed oscillator ready

Set and cleared by hardware to indicate when the internal RC 40 kHz oscillator is stable. After the LSION bit is cleared, LSIRDY goes low after 3 internal RC 40 kHz oscillator clock cycles.

0: Internal RC 40 kHz oscillator not ready

1: Internal RC 40 kHz oscillator ready

Bit 0 **LSION:** Internal low-speed oscillator enable

Set and cleared by software.

0: Internal RC 40 kHz oscillator OFF

1: Internal RC 40 kHz oscillator ON

6.3.11 RCC register map

The following table gives the RCC register map and the reset values.

Table 14. RCC - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	**RCC_CR**	Reserved						PLL RDY	PLL ON	Reserved				CSSON	HSEBYP	HSERDY	HSEON	HSICAL[7:0]								HSITRIM[4:0]					Reserved	HSIRDY	HSION
	Reset value							0	0					0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		1	1
0x004	**RCC_CFGR**	Reserved					MCO [2:0]			Reserved	USBPRE	PLLMUL[3:0]				PLLXTPRE	PLLSRC	ADC PRE [1:0]		PPRE2 [2:0]			PPRE1 [2:0]			HPRE[3:0]				SWS [1:0]		SW [1:0]	
	Reset value						0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x008	**RCC_CIR**	Reserved								CSSC	Reserved			PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved			PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	Reserved		PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF
	Reset value									0				0	0	0	0	0				0	0	0	0	0	0			0	0	0	0	0
0x00C	**RCC_APB2RSTR**	Reserved																ADC3RST	USART1RST	TIM8RST	SPI1RST	TIM1RST	ADC2RST	ADC1RST	IOPGRST	IOPFRST	IOPERST	IOPDRST	IOPCRST	IOPBRST	IOPARST	Reserved	AFIORST
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
0x010	**RCC_APB1RSTR**	Reserved	DACRST	PWRRST	BKPRST	Reserved	CANRST	Reserved		USBRST	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	Reserved	SPI3RST	SPI2RST	Reserved		WWDGRST	Reserved					TM7RST	TM6RST	TM5RST	TIM4RST	TIM3RST	TIM2RST
	Reset value		0	0	0		0			0	0	0	0	0	0	0		0	0			0						0	0	0	0	0	0
0x014	**RCC_AHBENR**	Reserved																					SDIOEN	Reserved	FSMCEN	Reserved	CRCEN	Reserved	FLITFEN	Reserved	SRAMEN	DM2AEN	DM1AEN
	Reset value																						0		0		0		1		1	0	0
0x018	**RCC_APB2ENR**	Reserved																ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Reserved	AFIOEN
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0

Table 14. RCC - register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x01C	**RCC_APB1ENR**	Reserved		DACEN	PWREN	BKPEN	Reserved	CANEN	Reserved	USBEN	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	Reserved	SPI3EN	SPI2EN	Reserved		WWDGEN	Reserved					TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN
	Reset value			0	0	0		0		0	0	0	0	0	0	0		0	0			0						0	0	0	0	0	0
0x020	**RCC_BDCR**	Reserved															BDRST	RTCEN	Reserved					RTC SEL [1:0]		Reserved					LSEBYP	LSERDY	LSEON
	Reset value																0	0						0	0						0	0	0
0x024	**RCC_CSR**	LPWRSTF	WWDGRSTF	IWDGRSTF	SFTRSTF	PORRSTF	PINRSTF	Reserved	RMVF	Reserved																						LSIRDY	LSION
	Reset value	0	0	0	0	1	1		0																							0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

7 General-purpose and alternate-function I/Os (GPIOs and AFIOs)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

7.1 GPIO functional description

Each of the general-purpose I/O ports has two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH), two 32-bit data registers (GPIOx_IDR, GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 16-bit reset register (GPIOx_BRR) and a 32-bit locking register (GPIOx_LCKR).

Subject to the specific hardware characteristics of each I/O port listed in the *datasheet*, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

- Input floating
- Input pull-up
- Input-pull-down
- Analog Input
- Output open-drain
- Output push-pull
- Alternate function push-pull
- Alternate function open-drain

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed). The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.

Figure 9 shows the basic structure of an I/O Port bit.

Figure 9. Basic structure of a standard I/O port bit

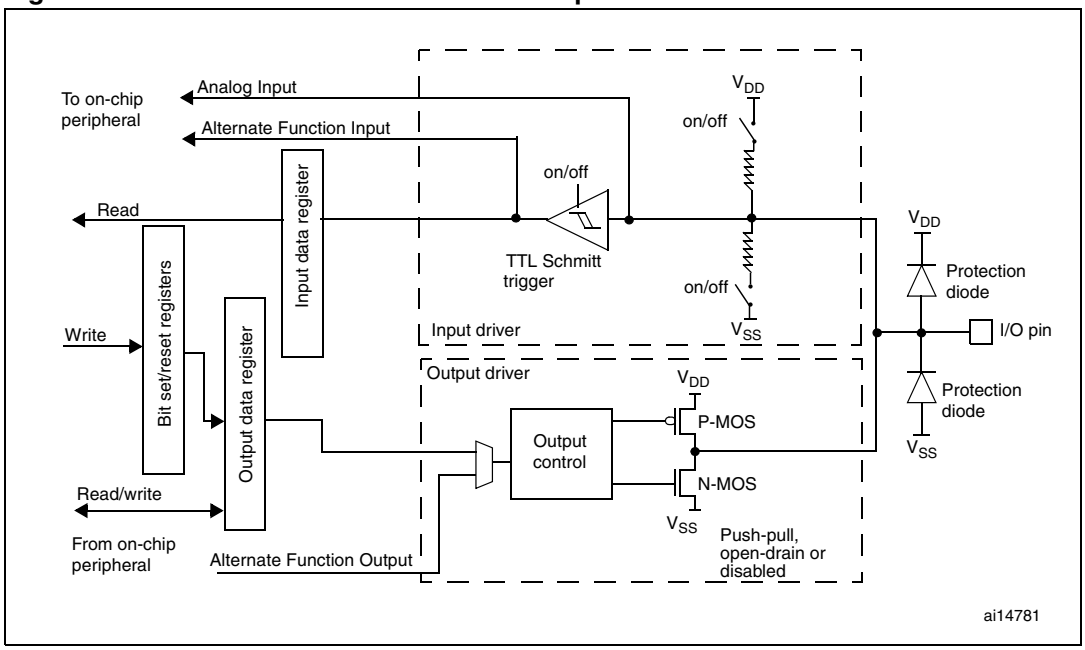
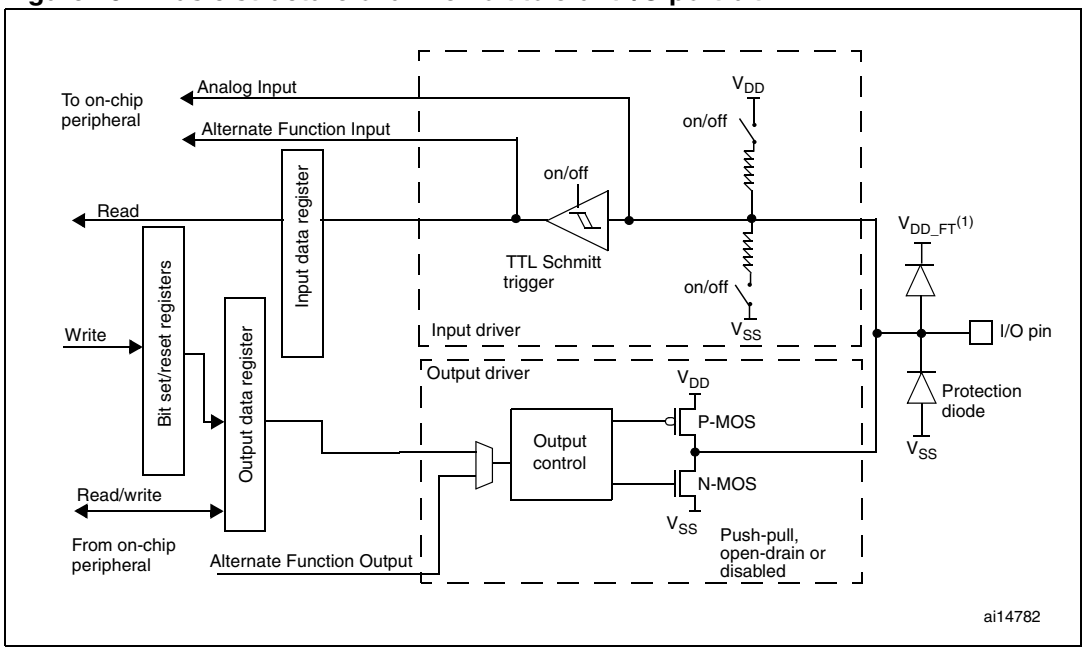


Figure 10. Basic structure of a five-volt tolerant I/O port bit



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

Table 15. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR Register
General purpose output	Push-pull	0	0	01 10 11 see *Table 16*		0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			don't care
	Open-drain		1			don't care
Input	Analog input	0	0	00		don't care
	Input floating		1			don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 16. Output MODE bits

MODE[1:0]	Meaning
00	Reserved
01	Max. output speed 10 MHz
10	Max. output speed 2 MHz
11	Max. output speed 50 MHz

7.1.1 General-purpose I/O (GPIO)

During and just after reset, the alternate functions are not active and the I/O ports are configured in Input Floating mode (CNFx[1:0]=01b, MODEx[1:0]=00b).

The JTAG pins are in input PU/PD after reset:

PA15: JTDI in PU

PA14: JTCK in PD

PA13: JTMS in PU

PB4: JNTRST in PU

When configured as output, the value written to the Output Data register (GPIOx_ODR) is output on the I/O pin. It is possible to use the output driver in Push-Pull mode or Open-Drain mode (only the N-MOS is activated when outputting 0).

The Input Data register (GPIOx_IDR) captures the data present on the I/O pin at every APB2 clock cycle.

All GPIO pins have a internal weak pull-up and weak pull-down which can be activated or not when configured as input.

7.1.2 Atomic bit set or reset

There is no need for the software to disable interrupts when programming the GPIOx_ODR at bit level: it is possible to modify only one or several bits in a single atomic APB2 write access. This is achieved by programming to '1' the Bit Set/Reset Register (GPIOx_BSRR, or for reset only GPIOx_BRR) to select the bits you want to modify. The unselected bits will not be modified.

7.1.3 External interrupt/wakeup lines

All ports have external interrupt capability. To use external interrupt lines, the port must be configured in input mode. For more information on external interrupts, refer to:

- *Section 8.2: External interrupt/event controller (EXTI) on page 120* and
- *Section 8.2.3: Wakeup event management on page 121*.

7.1.4 Alternate functions (AF)

It is necessary to program the Port Bit Configuration Register before using a default alternate function.

- For alternate function inputs, the port must be configured in Input mode (floating, pull-up or pull-down) and the input pin must be driven externally.

Note: It is also possible to emulate the AFI input pin by software by programming the GPIO controller. In this case, the port should be configured in Alternate Function Output mode. And obviously, the corresponding port should not be driven externally as it will be driven by the software using the GPIO controller.

- For alternate function outputs, the port must be configured in Alternate Function Output mode (Push-Pull or Open-Drain).
- For bidirectional Alternate Functions, the port bit must be configured in Alternate Function Output mode (Push-Pull or Open-Drain). In this case the input driver is configured in input floating mode

If you configure a port bit as Alternate Function Output, this disconnects the output register and connects the pin to the output signal of an on-chip peripheral.

If software configures a GPIO pin as Alternate Function Output, but peripheral is not activated, its output is not specified.

7.1.5 Software remapping of I/O alternate functions

To optimize the number of peripheral I/O functions for different device packages, it is possible to remap some alternate functions to some other pins. This is achieved by software, by programming the corresponding registers (refer to *AFIO registers on page 110*. In that case, the alternate functions are no longer mapped to their original assignations.

7.1.6 GPIO locking mechanism

The locking mechanism allows the IO configuration to be frozen. When the LOCK sequence has been applied on a port bit, it is no longer possible to modify the value of the port bit until the next reset.

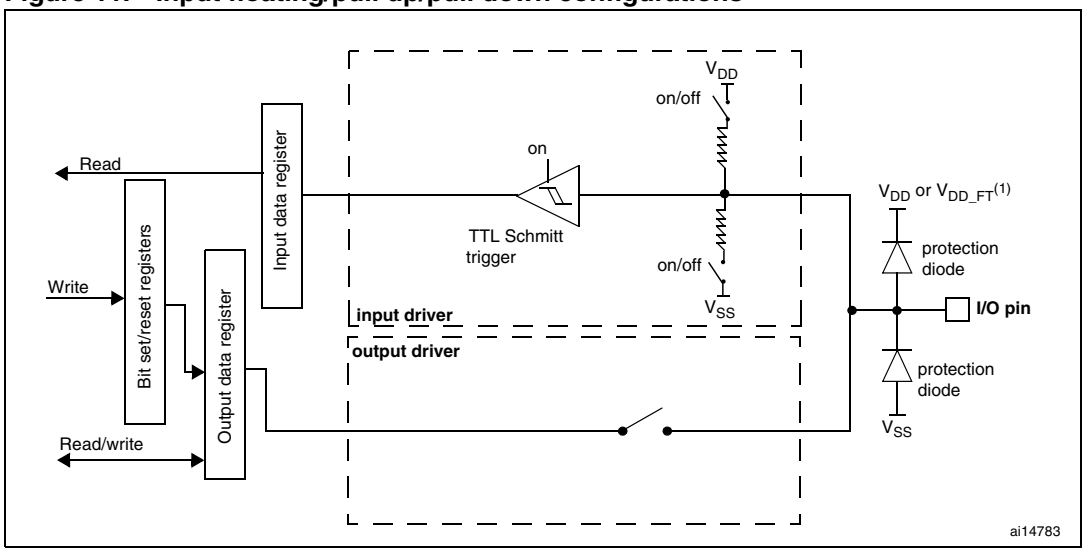
7.1.7 Input configuration

When the I/O Port is programmed as Input:

- The Output Buffer is disabled
- The Schmitt Trigger Input is activated
- The weak pull-up and pull-down resistors are activated or not depending on input configuration (pull-up, pull-down or floating):
- The data present on the I/O pin is sampled into the Input Data Register every APB2 clock cycle
- A read access to the Input Data Register obtains the I/O State.

The *Figure 11 on page 98* shows the Input Configuration of the I/O Port bit.

Figure 11. Input floating/pull up/pull down configurations



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

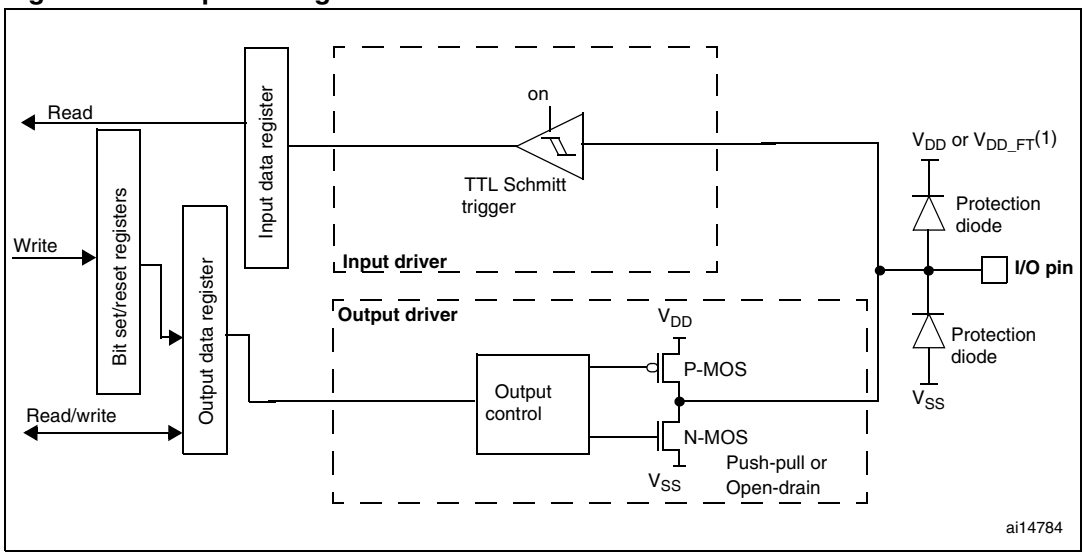
7.1.8 Output configuration

When the I/O Port is programmed as Output:

- The Output Buffer is enabled:
  - Open Drain Mode: A “0” in the Output register activates the N-MOS while a “1” in the Output register leaves the port in Hi-Z. (the P-MOS is never activated)
  - Push-Pull Mode: A “0” in the Output register activates the N-MOS while a “1” in the Output register activates the P-MOS.
- The Schmitt Trigger Input is activated.
- The weak pull-up and pull-down resistors are disabled.
- The data present on the I/O pin is sampled into the Input Data Register every APB2 clock cycle
- A read access to the Input Data Register gets the I/O state in open drain mode
- A read access to the Output Data register gets the last written value in Push-Pull mode

The *Figure 12 on page 99* shows the Output configuration of the I/O Port bit.

Figure 12. Output configuration



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

7.1.9 Alternate function configuration

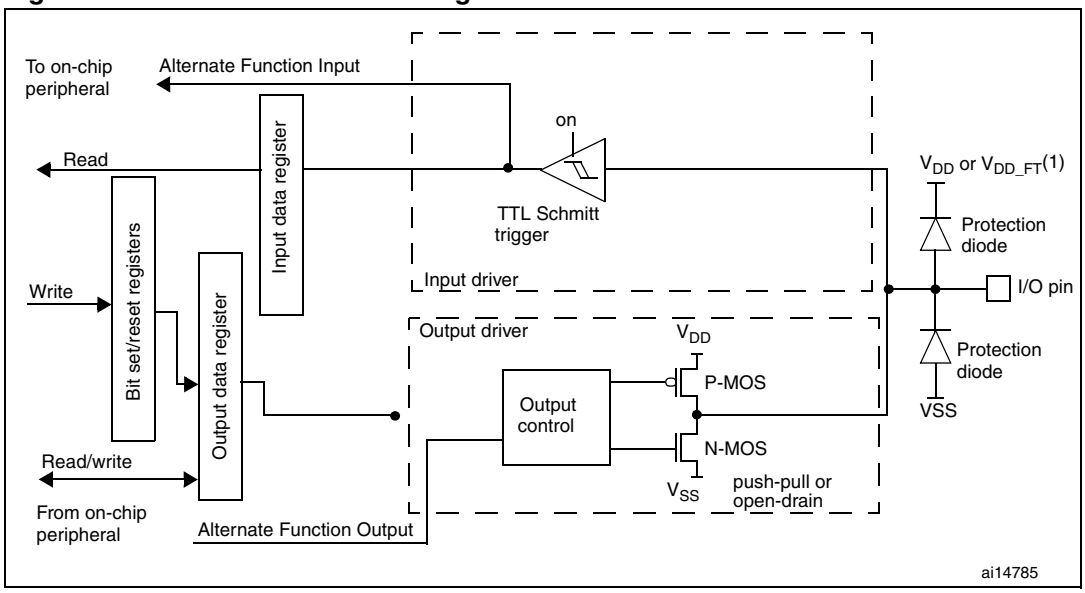
When the I/O Port is programmed as Alternate Function:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The Output Buffer is driven by the signal coming from the peripheral (alternate function out)
- The Schmitt Trigger Input is activated
- The weak pull-up and pull-down resistors are disabled.
- The data present on the I/O pin is sampled into the Input Data Register every APB2 clock cycle
- A read access to the Input Data Register gets the I/O state in open drain mode
- A read access to the Output Data register gets the last written value in Push-Pull mode

The *Figure 13 on page 100* shows the Alternate Function Configuration of the I/O Port bit. Also, refer to *Section 7.4: AFIO registers on page 110* for further information.

A set of Alternate Function I/O registers allow you to remap some alternate functions to different pins. Refer to

Figure 13. Alternate function configuration



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

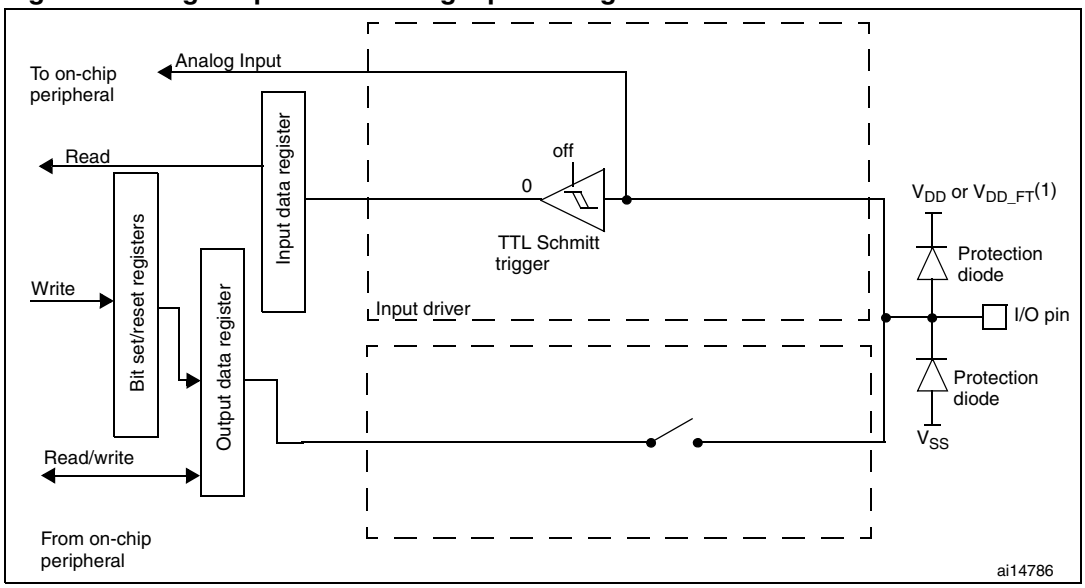
7.1.10 Analog input configuration

When the I/O Port is programmed as Analog Input Configuration:

- The Output Buffer is disabled.
- The Schmitt Trigger Input is de-activated providing zero consumption for every analog value of the I/O pin. The output of the Schmitt Trigger is forced to a constant value (0).
- The weak pull-up and pull-down resistors are disabled.
- Read access to the Input Data Register gets the value “0”.

The *Figure 14 on page 100* shows the High impedance-Analog Input Configuration of the I/O Port bit.

Figure 14. High impedance-analog input configuration



7.2 GPIO registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

7.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2

CNFy[1:0]: Port x configuration bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.

Refer to *Table 15: Port bit configuration table on page 96*.

In input mode (MODE[1:0]=00):

00: Analog input mode

01: Floating input (reset state)

10: Input with pull-up / pull-down

11: Reserved

In output mode (MODE[1:0] > 00):

00: General purpose output push-pull

01: General purpose output Open-drain

10: Alternate function output Push-pull

11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0

MODEy[1:0]: Port x mode bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.

Refer to *Table 15: Port bit configuration table on page 96*.

00: Input mode (reset state)

01: Output mode, max speed 10 MHz.

10: Output mode, max speed 2 MHz.

11: Output mode, max speed 50 MHz.

7.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.

Refer to *Table 15: Port bit configuration table on page 96*.

In input mode (MODE[1:0]=00):

00: Analog input mode

01: Floating input (reset state)

10: Input with pull-up / pull-down

11: Reserved

In output mode (MODE[1:0] > 00):

00: General purpose output push-pull

01: General purpose output Open-drain

10: Alternate function output Push-pull

11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.

Refer to *Table 15: Port bit configuration table on page 96*.

00: Input mode (reset state)

01: Output mode, max speed 10 MHz.

10: Output mode, max speed 2 MHz.

11: Output mode, max speed 50 MHz.

7.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Address offset: 0x08h

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, always read as 0.

Bits 15:0 **IDRy[15:0]:** Port input data (y= 0 .. 15)

These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

7.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, always read as 0.

Bits 15:0 **ODRy[15:0]:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

7.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

7.2.6 Port bit reset register (GPIOx_BRR) (x=A..G)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved

Bits 15:0 **BRy:** Port x Reset bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit

7.2.7 Port configuration lock register (GPIOx_LCKR) (x=A..G)

This register is used to lock the configuration of the port bits when a correct write sequence is applied to bit 16 (LCKK). The value of bits [15:0] is used to lock the configuration of the GPIO. During the write sequence, the value of LCKR[15:0] must not change. When the LOCK sequence has been applied on a port bit it is no longer possible to modify the value of the port bit until the next reset.

Each lock bit freezes the corresponding 4 bits of the control register (CRL, CRH).

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															LCKK
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved

Bit 16 **LCKK[16]:** Lock key

This bit can be read anytime. It can only be modified using the Lock Key Writing Sequence.
0: Port configuration lock key not active
1: Port configuration lock key active. GPIOx_LCKR register is locked until an MCU reset occurs.

LOCK key writing sequence:
Write 1
Write 0
Write 1
Read 0
Read 1 (this read is optional but confirms that the lock is active)

Note: During the LOCK Key Writing sequence, the value of LCK[15:0] must not change.
Any error in the lock sequence will abort the lock.

Bits 15:0 **LCKy:** Port x Lock bit y (y= 0 .. 15)

These bits are read write but can only be written when the LCKK bit is 0.
0: Port configuration not locked
1: Port configuration locked.

7.3 Alternate function I/O and debug configuration (AFIO)

To optimize the number of peripherals available for the 64-pin or the 100-pin or the 144-pin package, it is possible to remap some alternate functions to some other pins. This is achieved by software, by programming the *AF remap and debug I/O configuration register (AFIO_MAPR) on page 111*. In this case, the alternate functions are no longer mapped to their original assignations.

7.3.1 Using OSC32_IN/OSC32_OUT pins as GPIO ports PC14/PC15

The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general-purpose I/O PC14 and PC15, respectively, when the LSE oscillator is off. The LSE has priority over the GP IOs function.

Note: 1 The PC14/PC15 GPIO functionality is lost when the 1.8 V domain is powered off (by entering standby mode) or when the backup domain is supplied by V_{BAT} (V_{DD} no more supplied). In this case the IOs are set in analog input mode.

2 Refer to the note on IO usage restrictions in Section 4.1.2 on page 46.

7.3.2 Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1

The HSE oscillator pins OSC_IN/OSC_OUT can be used as general-purpose I/O PD0/PD1 by programming the PD01_REMAP bit in the *AF remap and debug I/O configuration register (AFIO_MAPR)*.

This remap is available only on 36-, 48- and 64-pin packages (PD0 and PD1 are available on 100-pin and 144-pin packages, no need for remapping).

Note: The external interrupt/event function is not remapped. PD0 and PD1 cannot be used for external interrupt/event generation on 36-, 48- and 64-pin packages.

7.3.3 CAN alternate function remapping

The CAN signals can be mapped on Port A, Port B or Port D as shown in *Table 17*. For port D, remapping is not possible in devices delivered in 36-, 48- and 64-pin packages.

Table 17. CAN1 alternate function remapping

Alternate function	CAN_REMAP[1:0] = “00”	CAN_REMAP[1:0] = “10” (1)	CAN_REMAP[1:0] = “11”(2)
CAN_RX	PA11	PB8	PD0
CAN_TX	PA12	PB9	PD1

1. Remap not available on 36-pin package
2. This remapping is available only on 100-pin and 144-pin packages, when PD0 and PD1 are not remapped on OSC-IN and OSC-OUT.

7.3.4 JTAG/SWD alternate function remapping

The debug interface signals are mapped on the GPIO ports as shown in *Table 18*.

Table 18. Debug interface signals

Alternate function	GPIO port
JTMS / SWDIO	PA13
JTCK / SWCLK	PA14
JTDI	PA15
JTDO / TRACESWO	PB3
JNTRST	PB4
TRACECK	PE2
TRACED0	PE3
TRACED1	PE4
TRACED2	PE5
TRACED3	PE6

To optimize the number of free GPIOs during debugging, this mapping can be configured in different ways by programming the SWJ_CFG[1:0] bits in the *AF remap and debug I/O configuration register (AFIO_MAPR)*. Refer to *Table 19*

Table 19. Debug port mapping

SWJ_CFG [2:0]	Available debug ports	SWJ I/O pin assigned				
		PA13 / JTMS/ SWDIO	PA14 / JTCK/S WCLK	PA15 / JTDI	PB3 / JTDO/ TRACE SWO	PB4/ JNTRST
000	Full SWJ (JTAG-DP + SW-DP) (Reset state)	X	X	X	X	X
001	Full SWJ (JTAG-DP + SW-DP) but without JNTRST	X	X	X	x	free
010	JTAG-DP Disabled and SW-DP Enabled	X	X	free	free(1)	free
100	JTAG-DP Disabled and SW-DP Disabled	free	free	free	free	free
Other	Forbidden					

1. Released only if not using asynchronous trace.

7.3.5 ADC alternate function remapping

Refer to *AF remap and debug I/O configuration register (AFIO_MAPR)*.

Table 20. ADC1 external trigger injected conversion alternate function remapping(1)

Alternate function	ADC1_ETRGINJ_REMAP = 0	ADC1_ETRGINJ_REMAP = 1
ADC1 external trigger injected conversion	ADC1 external trigger injected conversion is connected to EXTI15	ADC1 external trigger injected conversion is connected to TIM8_CH4

1. Remap available only for high-density devices.

Table 21. ADC1 external trigger regular conversion alternate function remapping(1)

Alternate function	ADC1_ETRGREG_REMAP = 0	ADC1_ETRGREG_REMAP = 1
ADC1 external trigger regular conversion	ADC1 external trigger regular conversion is connected to EXTI11	ADC1 external trigger regular conversion is connected to TIM8_TRGO

1. Remap available only for high-density devices.

Table 22. ADC2 external trigger injected conversion alternate function remapping(1)

Alternate function	ADC2_ETRGINJ_REMAP = 0	ADC2_ETRGINJ_REMAP = 1
ADC2 external trigger injected conversion	ADC2 external trigger injected conversion is connected to EXTI 15	ADC2 external trigger injected conversion is connected to TIM8_CH4

1. Remap available only for high-density devices.

Table 23. ADC2 external trigger regular conversion alternate function remapping(1)

Alternate function	ADC2_ETRGREG_REG = 0	ADC2_ETRGREG_REG = 1
ADC2 external trigger regular conversion	ADC2 external trigger regular conversion is connected to EXTI11	ADC2 external trigger regular conversion is connected to TIM8_TRGO

1. Remap available only for high-density devices.

7.3.6 Timer alternate function remapping

Timer 4 channels 1 to 4 can be remapped from Port B to Port D. Other timer remapping possibilities are listed in *Table 26* to *Table 28*. Refer to *AF remap and debug I/O configuration register (AFIO_MAPR)*.

Table 24. Timer 5 alternate function remapping(1)

Alternate function	TIM5CH4_IREMAP = 0	TIM5CH4_IREMAP = 1
TIM5_CH4	TIM5 Channel4 is connected to PA3	LSI internal clock is connected to TIM5_CH4 input for calibration purpose.

1. Remap available only for high-density devices.

Table 25. Timer 4 alternate function remapping

Alternate function	TIM4_REMAP = 0	TIM4_REMAP = 1[1]
TIM4_CH1	PB6	PD12
TIM4_CH2	PB7	PD13
TIM4_CH3	PB8	PD14
TIM4_CH4	PB9	PD15

1. Remap available only for 100-pin and for 144-pin package.

Table 26. Timer 3 alternate function remapping

Alternate function	TIM3_REMAP[1:0] = “00” (no remap)	TIM3_REMAP[1:0] = “10” (partial remap)	TIM3_REMAP[1:0] = “11” (full remap) [1]
TIM3_CH1	PA6	PB4	PC6
TIM3_CH2	PA7	PB5	PC7
TIM3_CH3	PB0		PC8
TIM3_CH4	PB1		PC9

1. Remap available only for 64-pin, 100-pin and 144-pin packages.

Table 27. Timer 2 alternate function remapping

Alternate function	TIM2_REMAP[1:0] = “00” (no remap)	TIM2_REMAP[1:0] = “01” (partial remap)	TIM2_REMAP[1:0] = “10” (partial remap) [1]	TIM2_REMAP[1:0] = “11” (full remap) [1]
TIM2_CH1_ETR[2]	PA0	PA15	PA0	PA15
TIM2_CH2	PA1	PB3	PA1	PB3
TIM2_CH3	PA2		PB10	
TIM2_CH4	PA3		PB11	

1. Remap not available on 36-pin package.
2. TIM_CH1 and TIM_ETR share the same pin but cannot be used at the same time (which is why we have this notation: TIM2_CH1_ETR).

Table 28. Timer 1 alternate function remapping

Alternate functions mapping	TIM1_REMAP[1:0] = "00" (no remap)	TIM1_REMAP[1:0] = "01" (partial remap)	TIM1_REMAP[1:0] = "11" (full remap) (1)
TIM1_ETR	PA12		PE7
TIM1_CH1	PA8		PE9
TIM1_CH2	PA9		PE11
TIM1_CH3	PA10		PE13
TIM1_CH4	PA11		PE14
TIM1_BKIN	PB12 (2)	PA6	PE15
TIM1_CH1N	PB13 (2)	PA7	PE8
TIM1_CH2N	PB14 (2)	PB0	PE10
TIM1_CH3N	PB15 (2)	PB1	PE12

1. Remap available only for 100-pin and 144-pin packages.
2. Remap not available on 36-pin package.

7.3.7 USART Alternate function remapping

Refer to *AF remap and debug I/O configuration register (AFIO_MAPR)*.

Table 29. USART3 remapping

Alternate function	USART3_REMAP[1:0] = "00" (no remap)	USART3_REMAP[1:0] = "01" (partial remap) (1)	USART3_REMAP[1:0] = "11" (full remap) (2)
USART3_TX	PB10	PC10	PD8
USART3_RX	PB11	PC11	PD9
USART3_CK	PB12	PC12	PD10
USART3_CTS	PB13		PD11
USART3_RTS	PB14		PD12

1. Remap available only for 64-pin, 100-pin and 144-pin packages
2. Remap available only for 100-pin and 144-pin packages.

Table 30. USART2 remapping

Alternate functions	USART2_REMAP = 0	USART2_REMAP = 1 (1)
USART2_CTS	PA0	PD3
USART2_RTS	PA1	PD4
USART2_TX	PA2	PD5
USART2_RX	PA3	PD6
USART2_CK	PA4	PD7

1. Remap available only for 100-pin and 144-pin packages.

Table 31. USART1 remapping

Alternate function	USART1_REMAP = 0	USART1_REMAP = 1
USART1_TX	PA9	PB6
USART1_RX	PA10	PB7

7.3.8 I2C 1 alternate function remapping

Refer to *AF remap and debug I/O configuration register (AFIO_MAPR)*

Table 32. I2C1 remapping

Alternate function	I2C1_REMAP = 0	I2C1_REMAP = 1 (1)
I2C1_SCL	PB6	PB8
I2C1_SDA	PB7	PB9

1. Remap not available on 36-pin package.

7.3.9 SPI 1 alternate function remapping

Refer to *AF remap and debug I/O configuration register (AFIO_MAPR)*

Table 33. SPI1 remapping

Alternate function	SPI1_REMAP = 0	SPI1_REMAP = 1
SPI1_NSS	PA4	PA15
SPI1_SCK	PA5	PB3
SPI1_MISO	PA6	PB4
SPI1_MOSI	PA7	PB5

7.4 AFIO registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

Note: To read/write the AFIO_EVCR,AFIO_MAPR and AFIO_EXTICRX registers, the AFIO clock should first be enabled. Refer to Section 6.3.7: APB2 peripheral clock enable register (RCC_APB2ENR).

7.4.1 Event control register (AFIO_EVCR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EVOE	PORT[2:0]			PIN[3:0]			
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved

Bit 7 **EVOE:** Event output enable

Set and cleared by software. When set the EVENTOUT Cortex output is connected to the I/O selected by the PORT[2:0] and PIN[3:0] bits.

Bits 6:4 **PORT[2:0]**: Port selection

Set and cleared by software. Select the port used to output the Cortex EVENTOUT signal.

Note: The EVENTOUT signal output capability is not extended to ports PF and PG.

000: PA selected
001: PB selected
010: PC selected
011: PD selected
100: PE selected

Bits 3:0 **PIN[3:0]:** Pin selection (x = A .. E)

Set and cleared by software. Select the pin used to output the Cortex EVENTOUT signal.

0000: Px0 selected
0001: Px1 selected
0010: Px2 selected
0011: Px3 selected
...
1111: Px15 selected

7.4.2 AF remap and debug I/O configuration register (AFIO_MAPR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SWJ_CFG[2:0]			Reserved			ADC2_ETRGREG_REMAP	ADC2_ETRGINJ_REMAP	ADC1_ETRGREG_REMAP	ADC1_ETRGINJ_REMAP	TIM5CH4_IREMAP
					w	w	w				rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD01_REMAP	CAN1_REMAP [1:0]		TIM4_REMAP	TIM3_REMAP [1:0]		TIM2_REMAP [1:0]		TIM1_REMAP [1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:27 Reserved

Bits 26:24 **SWJ_CFG[2:0]** Serial wire JTAG configuration

These bits are write-only (when read, the value is undefined). They are used to configure the SWJ and trace alternate function I/Os. The SWJ (Serial Wire JTAG) supports JTAG or SWD access to the Cortex debug port. The default state after reset is SWJ ON without trace. This allows JTAG or SW mode to be enabled by sending a specific sequence on the JTMS / JTCK pin.

000: Full SWJ (JTAG-DP + SW-DP): Reset State
001: Full SWJ (JTAG-DP + SW-DP) but without JNTRST
010: JTAG-DP Disabled and SW-DP Enabled
100: JTAG-DP Disabled and SW-DP Disabled
Other combinations: no effect

Bits 23:21 Reserved

Bits 20 **ADC2_ETRGREG_REMAP:** ADC 2 external trigger regular conversion remapping

Set and cleared by software. This bit controls the trigger input connected to ADC2 external trigger regular conversion. When this bit is reset, the ADC2 external trigger regular conversion is connected to EXTI11. When this bit is set, the ADC2 external event regular conversion is connected to TIM8_TRGO.

Bits 19 **ADC2_ETRGINJ_REMAP:** ADC 2 external trigger injected conversion remapping

Set and cleared by software. This bit controls the trigger input connected to ADC2 external trigger injected conversion. When this bit is reset, the ADC2 external trigger injected conversion is connected to EXTI15. When this bit is set, the ADC2 external event injected conversion is connected to TIM8_Channel4.

Bits 18 **ADC1_ETRGREG_REMAP:** ADC 1 external trigger regular conversion remapping

Set and cleared by software. This bit controls the trigger input connected to ADC1 External trigger regular conversion. When reset the ADC1 External trigger regular conversion is connected to EXTI11. When set the ADC1 External Event regular conversion is connected to TIM8 TRGO.

Bits 17 **ADC1_ETRGINJ_REMAP:** ADC 1 External trigger injected conversion remapping

Set and cleared by software. This bit controls the trigger input connected to ADC1 External trigger injected conversion. When reset the ADC1 External trigger injected conversion is connected to EXTI15. When set the ADC1 External Event injected conversion is connected to TIM8 Channel4.

Bits 16 **TIM5CH4_IREMAP:** TIM5 channel4 internal remap

Set and cleared by software. This bit controls the TIM5_CH4 internal mapping. When reset the timer TIM5_CH4 is connected to PA3. When set the LSI internal clock is connected to TIM5_CH4 input for calibration purpose.

Bit 15 **PD01_REMAP:** Port D0/Port D1 mapping on OSC_IN/OSC_OUT

This bit is set and cleared by software. It controls the mapping of PD0 and PD1 GPIO functionality. When the HSE oscillator is not used (application running on internal 8 MHz RC) PD0 and PD1 can be mapped on OSC_IN and OSC_OUT. This is available only on 36-, 48- and 64-pin packages (PD0 and PD1 are available on 100-pin and 144-pin packages, no need for remapping).

0: No remapping of PD0 and PD1

1: PD0 remapped on OSC_IN, PD1 remapped on OSC_OUT,

Bits 14:13 **CAN_REMAP[1:0]**: CAN alternate function remapping

These bits are set and cleared by software. They control the mapping of Alternate Functions CAN_RX and CAN_TX.

00: CAN_RX mapped to PA11, CAN_TX mapped to PA12

01: Not used

10: CAN_RX mapped to PB8, CAN_TX mapped to PB9 (not available on 36-pin package)

11: CAN_RX mapped to PD0, CAN_TX mapped to PD1

Bit 12 **TIM4_REMAP:** TIM4 remapping

This bit is set and cleared by software. It controls the mapping of TIM4 channels 1 to 4 onto the GPIO ports.

0: No remap (TIM4_CH1/PB6, TIM4_CH2/PB7, TIM4_CH3/PB8, TIM4_CH4/PB9)

1: Full remap (TIM4_CH1/PD12, TIM4_CH2/PD13, TIM4_CH3/PD14, TIM4_CH4/PD15)

Note: TIM4_ETR on PE0 is not re-mapped.

Bits 11:10 **TIM3_REMAP[1:0]** TIM3 remapping

These bits are set and cleared by software. They control the mapping of TIM3 channels 1 to 4 on the GPIO ports.
00: No remap (CH1/PA6, CH2/PA7, CH3/PB0, CH4/PB1)
01: Not used
10: Partial remap (CH1/PB4, CH2/PB5, CH3/PB0, CH4/PB1)
11: Full remap (CH1/PC6, CH2/PC7, CH3/PC8, CH4/PC9)

Note: TIM3_ETR on PE0 is not re-mapped.

Bits 9:8 **TIM2_REMAP[1:0]** TIM2 remapping

These bits are set and cleared by software. They control the mapping of TIM2 channels 1 to 4 and external trigger (ETR) on the GPIO ports.
00: No remap (CH1/ETR/PA0, CH2/PA1, CH3/PA2, CH4/PA3)
01: Partial remap (CH1/ETR/PA15, CH2/PB3, CH3/PA2, CH4/PA3)
10: Partial remap (CH1/ETR/PA0, CH2/PA1, CH3/PB10, CH4/PB11)
11: Full remap (CH1/ETR/PA15, CH2/PB3, CH3/PB10, CH4/PB11)

Bits 7:6 **TIM1_REMAP[1:0]** TIM1 remapping

These bits are set and cleared by software. They control the mapping of TIM2 channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN) on the GPIO ports.
00: No remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PB12, CH1N/PB13, CH2N/PB14, CH3N/PB15)
01: Partial remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PA6, CH1N/PA7, CH2N/PB0, CH3N/PB1)
10: not used
11: Full remap (ETR/PE7, CH1/PE9, CH2/PE11, CH3/PE13, CH4/PE14, BKIN/PE15, CH1N/PE8, CH2N/PE10, CH3N/PE12)

Bits 5:4 **USART3_REMAP[1:0]** USART3 remapping

These bits are set and cleared by software. They control the mapping of USART3 CTS, RTS,CK,TX and RX alternate functions on the GPIO ports.
00: No remap (TX/PB10, RX/PB11, CK/PB12, CTS/PB13, RTS/PB14)
01: Partial remap (TX/PC10, RX/PC11, CK/PC12, CTS/PB13, RTS/PB14)
10: not used
11: Full remap (TX/PD8, RX/PD9, CK/PD10, CTS/PD11, RTS/PD12)

Bit 3 **USART2_REMAP** USART2 remapping

This bit is set and cleared by software. It controls the mapping of USART2 CTS, RTS,CK,TX and RX alternate functions on the GPIO ports.
0: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)
1: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)

Bit 2 **USART1_REMAP** USART1 remapping

This bit is set and cleared by software. It controls the mapping of USART1 TX and RX alternate functions on the GPIO ports.
0: No remap (TX/PA9, RX/PA10)
1: Remap (TX/PB6, RX/PB7)

Bit 1 **I2C1_REMAP** I2C1 remapping

This bit is set and cleared by software. It controls the mapping of I2C1 SCL and SDA alternate functions on the GPIO ports.
0: No remap (SCL/PB6, SDA/PB7)
1: Remap (SCL/PB8, SDA/PB9)

Bit 0 **SPI1_REMAP** SPI1 remapping

This bit is set and cleared by software. It controls the mapping of SPI1 NSS, SCK, MISO, MOSI alternate functions on the GPIO ports.
0: No remap (NSS/PA4, SCK/PA5, MISO/PA6, MOSI/PA7)
1: Remap (NSS/PA15, SCK/PB3, MISO/PB4, MOSI/PB5)

7.4.3 External interrupt configuration register 1 (AFIO_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x= 0 to 3)

These bits are written by software to select the source input for EXTIx external interrupt.
Refer to *Section 8.2.5: External interrupt/event line mapping on page 122*
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

7.4.4 External interrupt configuration register 2 (AFIO_EXTICR2)

Address offset: 0x0C
Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x= 4 to 7)

These bits are written by software to select the source input for EXTIx external interrupt.
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

7.4.5 External interrupt configuration register 3 (AFIO_EXTICR3)

Address offset: 0x10

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x= 8 to 11)

These bits are written by software to select the source input for EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

7.4.6 External interrupt configuration register 4 (AFIO_EXTICR4)

Address offset: 0x14

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x= 12 to 15)

These bits are written by software to select the source input for EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

7.5 GPIO and AFIO register maps

Refer to *Table 1 on page 35* for the register boundary addresses. The following tables give the GPIO and AFIO register map and the reset values.

Table 34. GPIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**GPIOx_CRL**	CNF 7 [1:0]		MODE 7 [1:0]		CNF 6 [1:0]		MODE 6 [1:0]		CNF 5 [1:0]		MODE 5 [1:0]		CNF 4 [1:0]		MODE 4 [1:0]		CNF 3 [1:0]		MODE 3 [1:0]		CNF 2 [1:0]		MODE 2 [1:0]		CNF 1 [1:0]		MODE 1 [1:0]		CNF 0 [1:0]		MODE 0 [1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x04	**GPIOx_CRH**	CNF 15 [1:0]		MODE 15 [1:0]		CNF 14 [1:0]		MODE 14 [1:0]		CNF 13 [1:0]		MODE 13 [1:0]		CNF 12 [1:0]		MODE 12 [1:0]		CNF 11 [1:0]		MODE 11 [1:0]		CNF 10 [1:0]		MODE 10 [1:0]		CNF 9 [1:0]		MODE 9 [1:0]		CNF 8 [1:0]		MODE 8 [1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x08	**GPIOx_IDR**	Reserved																IDR[15:0]I															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**GPIOx_ODR**	Reserved																ODR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**GPIOx_BSRR**	BR[15:0]																BSR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**GPIOx_BRR**	Reserved																BR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**GPIOx_LCKR**	Reserved															LCKK	LCK[15:0]															
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 35. AFIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**AFIO_EVCR**	Reserved																									EVOE	PORT[2:0]			PIN[3:0]			
	Reset value																										0	0	0	0	0	0	0
0x04	**AFIO_MAPR**	Reserved					SWJ_CFG[2]	SWJ_CFG[1]	SWJ_CFG[0]	Reserved			ADC2_ETRGREG_REMAP	ADC2_ETRGINJ_REMAP	ADC1_ETRGREG_REMAP	ADC1_ETRGINJ_REMAP	TIM5CH4_IREMAP	PD01_REMAP	CAN1_REMAP[1]	CAN1_REMAP[0]	TIM4_REMPAP	TIM3_REMPAP[1]	TIM3_REMPAP[0]	TIM2_REMPAP[1]	TIM2_REMPAP[0]	TIM1_REMPAP[1]	TIM1_REMPAP[0]	USART3_REMAP[1]	USART3_REMAP[0]	USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
	Reset value						0	0	0									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**AFIO_EXTICR1**	Reserved																EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**AFIO_EXTICR2**	Reserved																EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**AFIO_EXTICR3**	Reserved																EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**AFIO_EXTICR4**	Reserved																EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

8 Interrupts and events

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

8.1 Nested vectored interrupt controller (NVIC)

Features

- 60 maskable interrupt channels (not including the 16 interrupt lines of Cortex™-M3)
- 16 programmable priority levels (4 bits of interrupt priority are used)
- Low-latency exception and interrupt handling
- Power management control
- Implementation of System Control Registers

The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts.

All interrupts including the core exceptions are managed by the NVIC. For more information on exceptions and NVIC programming see Chap 5 Exceptions & Chap 8 Nested Vectored Interrupt Controller of the ARM *Cortex™-M3 Technical Reference Manual*.

8.1.1 SysTick calibration value register

The SysTick calibration value is fixed to 9000 which allows the generation of a time base of 1ms with the SysTick clock set to 9 MHz (max HCLK/8).

8.1.2 Interrupt and exception vectors

Table 36 is the vector table for STM32F10xxx devices.

Table 36. Vector table for other STM32F10xxx devices

Position	Priority	Type of priority	Acronym	Description	Address
	-	-	-	Reserved	0x0000_0000
	-3	fixed	Reset	Reset	0x0000_0004
	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
	-1	fixed	HardFault	All class of fault	0x0000_000C

Table 36. Vector table for other STM32F10xxx devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
	0	settable	MemManage	Memory management	0x0000_0010
	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
	-	-	-	Reserved	0x0000_001C - 0x0000_002B
	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
	-	-	-	Reserved	0x0000_0034
	5	settable	PendSV	Pendable request for system service	0x0000_0038
	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
19	26	settable	USB_HP_CAN_TX	USB High Priority or CAN TX interrupts	0x0000_008C
20	27	settable	USB_LP_CAN_RX0	USB Low Priority or CAN RX0 interrupts	0x0000_0090

Table 36. Vector table for other STM32F10xxx devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
21	28	settable	CAN_RX1	CAN RX1 interrupt	0x0000_0094
22	29	settable	CAN_SCE	CAN SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000_00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
42	49	settable	USBWakeup	USB wakeup from suspend through EXTI line interrupt	0x0000_00E8
43	50	settable	TIM8_BRK	TIM8 Break interrupt	0x0000_00EC
44	51	settable	TIM8_UP	TIM8 Update interrupt	0x0000_00F0
45	52	settable	TIM8_TRG_COM	TIM8 Trigger and Commutation interrupts	0x0000_00F4
46	53	settable	TIM8_CC	TIM8 Capture Compare interrupt	0x0000_00F8
47	54	settable	ADC3	ADC3 global interrupt	0x0000_00FC
48	55	settable	FSMC	FSMC global interrupt	0x0000_0100
49	56	settable	SDIO	SDIO global interrupt	0x0000_0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C

Table 36. Vector table for other STM32F10xxx devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
52	59	settable	UART4	UART4 global interrupt	0x0000_0110
53	60	settable	UART5	UART5 global interrupt	0x0000_0114
54	61	settable	TIM6	TIM6 global interrupt	0x0000_0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C
56	63	settable	DMA2_Channel1	DMA2 Channel1 global interrupt	0x0000_0120
57	64	settable	DMA2_Channel2	DMA2 Channel2 global interrupt	0x0000_0124
58	65	settable	DMA2_Channel3	DMA2 Channel3 global interrupt	0x0000_0128
59	66	settable	DMA2_Channel4_5	DMA2 Channel4 and DMA2 Channel5 global interrupts	0x0000_012C

8.2 External interrupt/event controller (EXTI)

The external interrupt/event controller consists of up to 19 edge detectors for generating event/interrupt requests. Each input line can be independently configured to select the type (pulse or pending) and the corresponding trigger event (rising or falling or both). Each line can also masked independently. A pending register maintains the status line of the interrupt requests

8.2.1 Main features

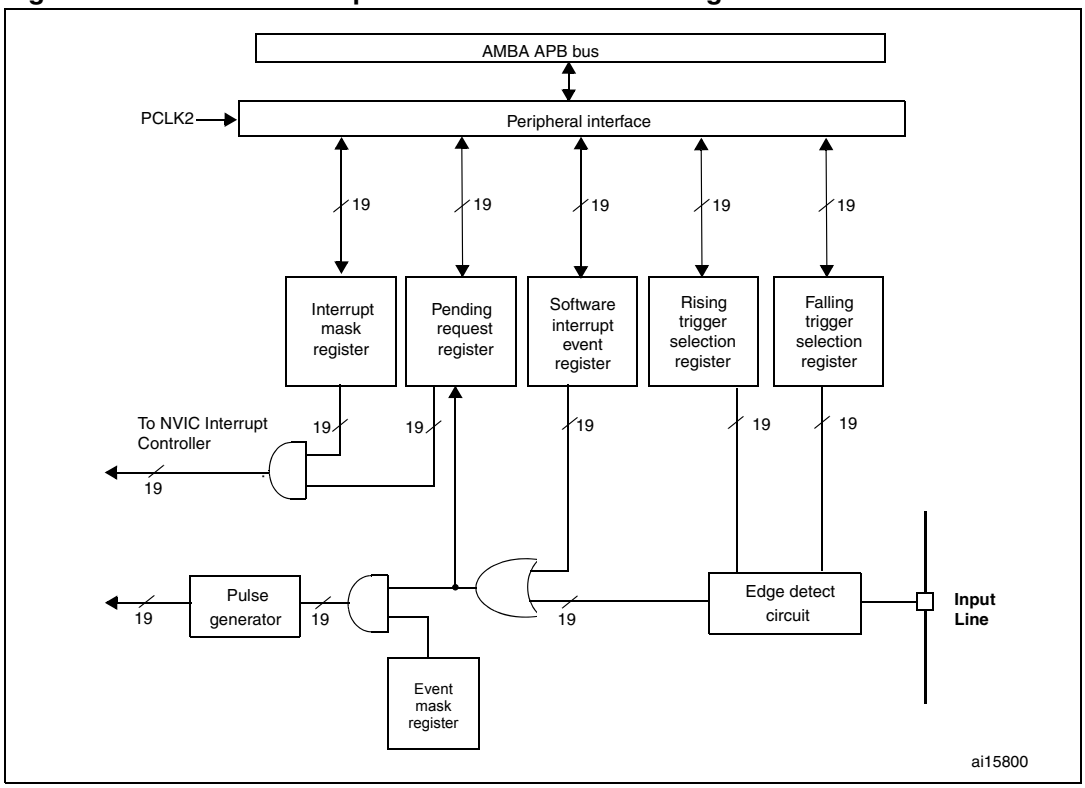
The EXTI controller main features are the following:

- Independent trigger and mask on each interrupt/event line
- Dedicated status bit for each interrupt line
- Generation of up to 19 software event/interrupt requests
- Detection of external signal with pulse width lower than APB2 clock period. Refer to the electrical characteristics section of the datasheet for details on this parameter.

8.2.2 Block diagram

The block diagram is shown in *Figure 15*.

Figure 15. External interrupt/event controller block diagram



8.2.3 Wakeup event management

The STM32F10xxx is able to handle external or internal events in order to wake up the core (WFE). The wakeup event can be generated either by:

- enabling an interrupt in the peripheral control register but not in the NVIC, and enabling the SEVONPEND bit in the Cortex-M3 System Control register. When the MCU resumes from WFE, the peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.
- or configuring an external or internal EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bit corresponding to the event line is not set.

To use an external line as a wakeup event, refer to *Section 8.2.4: Functional description*.

8.2.4 Functional description

To generate the interrupt, the interrupt line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the interrupt request by writing a ‘1’ to the corresponding bit in the interrupt mask register. When the selected edge occurs on the external interrupt line, an interrupt request is generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a ‘1’ in the pending register.

To generate the event, the event line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the event request by writing a ‘1’ to the corresponding bit in the event mask register. When the

selected edge occurs on the event line, an event pulse is generated. The pending bit corresponding to the event line is not set

An interrupt/event request can also be generated by software by writing a ‘1’ in the software interrupt/event register.

Hardware interrupt selection

To configure the 19 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 19 Interrupt lines (EXTI_IMR)
- Configure the Trigger Selection bits of the Interrupt lines (EXTI_RTSR and EXTI_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the External Interrupt Controller (EXTI) so that an interrupt coming from one of the 19 lines can be correctly acknowledged.

Hardware event selection

To configure the 19 lines as event sources, use the following procedure:

- Configure the mask bits of the 19 Event lines (EXTI_EMR)
- Configure the Trigger Selection bits of the Event lines (EXTI_RTSR and EXTI_FTSR)

Software interrupt/event selection

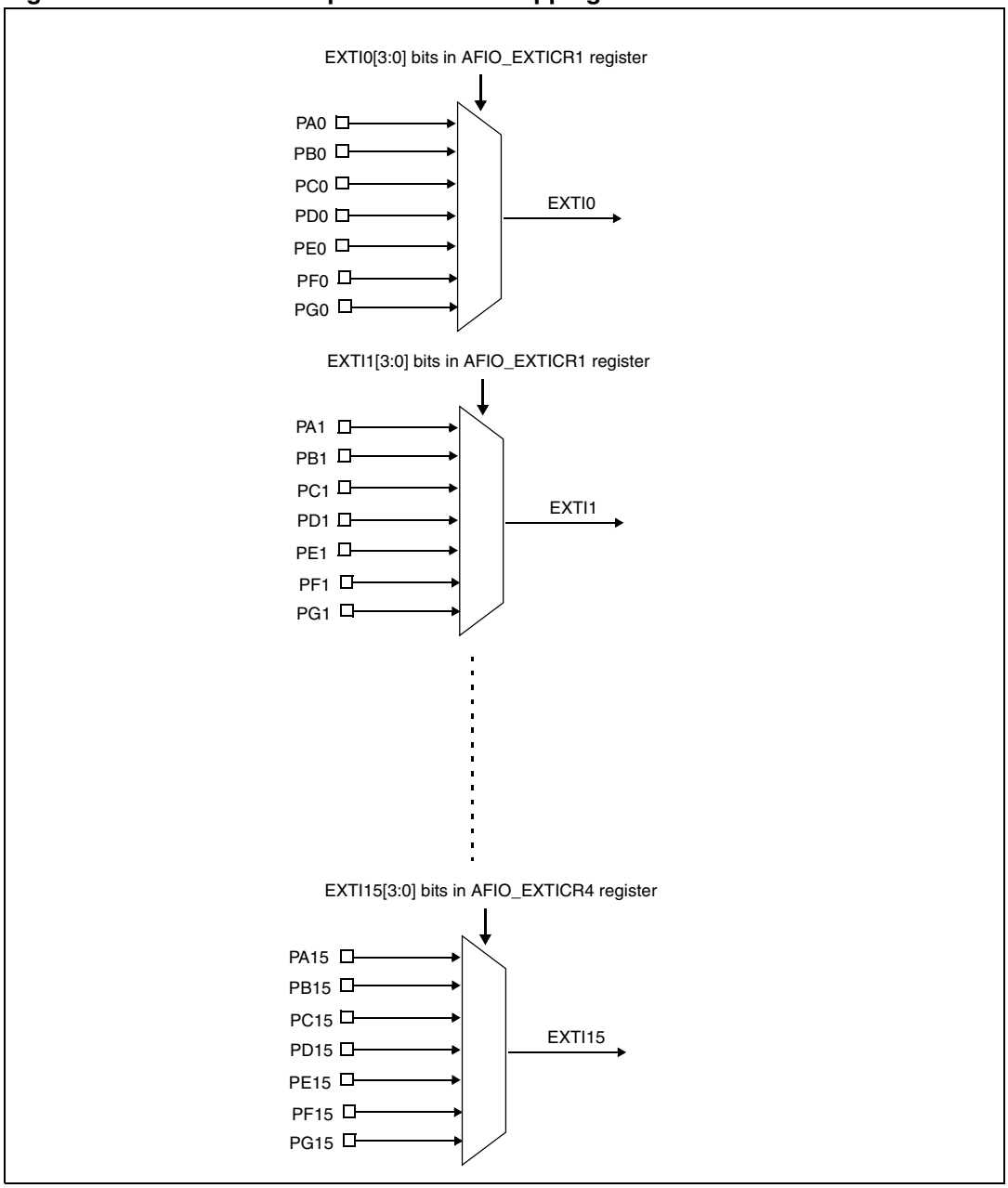
The 19 lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

- Configure the mask bits of the 19 Interrupt/Event lines (EXTI_IMR, EXTI_EMR)
- Set the required bit of the software interrupt register (EXTI_SWIER)

8.2.5 External interrupt/event line mapping

The 112 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

Figure 16. External interrupt/event GPIO mapping



The three other EXTI lines are connected as follows:

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB Wakeup event

8.3 EXTI registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

8.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													MR18	MR17	MR16
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **MRx:** Interrupt Mask on line x

Note: 0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked

8.3.2 Event mask register (EXTI_EMR)

Address offset: 0x04
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													MR18	MR17	MR16
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **MRx:** Event Mask on line x

Note: 0: Event request from Line x is masked
1: Event request from Line x is not masked

8.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TR18	TR17	TR16
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **TRx:** Rising trigger event configuration bit of line x

Note: 0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line.

Note: The external wakeup lines are edge triggered, no glitches must be generated on these lines. If a rising edge on external interrupt line occurs during writing of EXTI_RTSR register, the pending bit will not be set.

Rising and Falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

8.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TR18	TR17	TR16
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **TRx:** Falling trigger event configuration bit of line x

Note: 0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

Note: The external wakeup lines are edge triggered, no glitches must be generated on these lines. If a falling edge on external interrupt line occurs during writing of EXTI_FTSR register, the pending bit will not be set.

Rising and Falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

8.3.5 Software interrupt event register (EXTI_SWIER)

Address offset: 0x10
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													SWIER 18	SWIER 17	SWIER 16
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **SWIERx:** *Software Interrupt on line x*

Writing a 1 to this bit when it is at 0 sets the corresponding pending bit in EXTI_PR. If the interrupt is enabled on this line on the EXTI_IMR and EXTI_EMR, an interrupt request is generated.

Note: This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a 1 into the bit).

8.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													PR18	PR17	PR16
													rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **PRx:** *Pending bit*

Note: 0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a 1 into the bit or by changing the sensitivity of the edge detector.

8.3.7 EXTI register map

The following table gives the EXTI register map and the reset values.

Table 37. External interrupt/event controller register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**EXTI_IMR**	Reserved													MR[18:0]																		
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	**EXTI_EMR**	Reserved													MR[18:0]																		
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**EXTI_RTSR**	Reserved													TR[18:0]																		
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**EXTI_FTSR**	Reserved													TR[18:0]																		
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**EXTI_SWIER**	Reserved													SWIER[18:0]																		
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**EXTI_PR**	Reserved													PR[18:0]																		
	Reset value														x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Refer to *Table 1 on page 35* for the register boundary addresses.

9 DMA controller (DMA)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

9.1 DMA introduction

Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. This keeps CPU resources free for other operations.

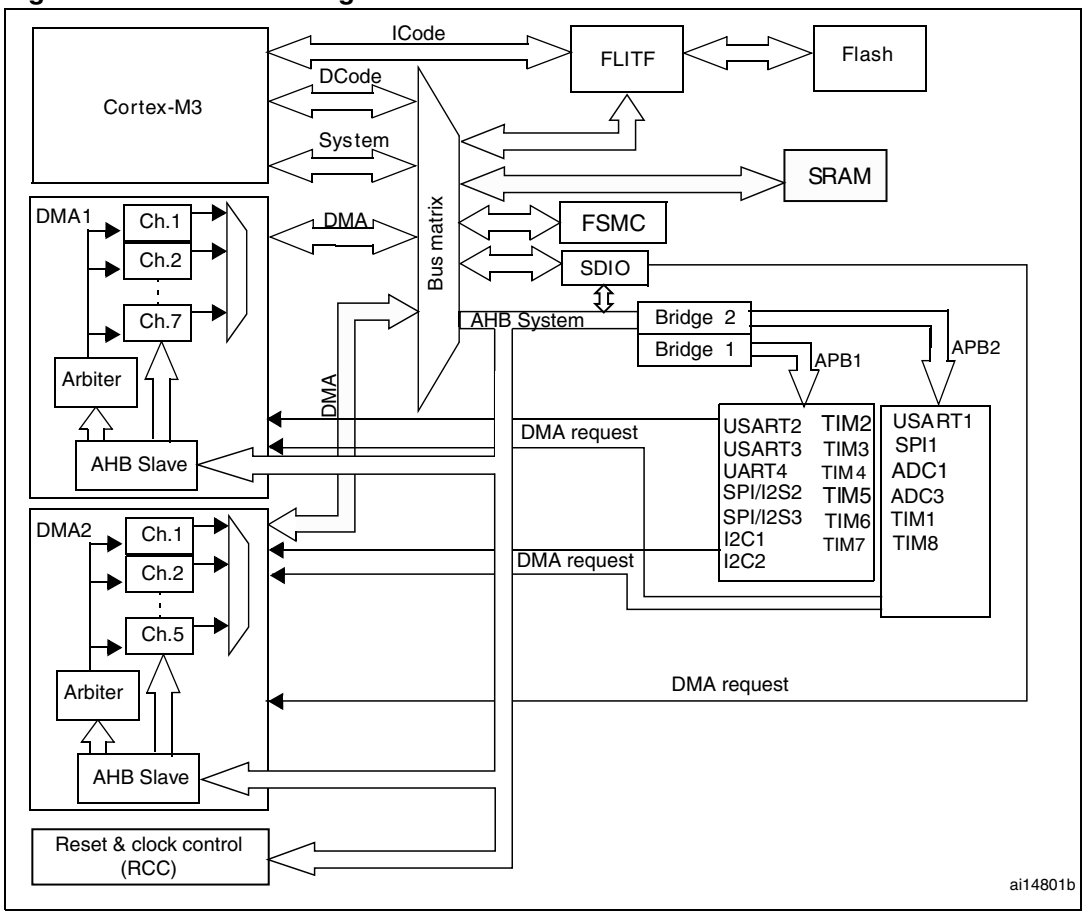
The two DMA controllers have 12 channels in total (7 for DMA1 and 5 for DMA2), each dedicated to managing memory access requests from one or more peripherals. It has an arbiter for handling the priority between DMA requests.

9.2 DMA main features

- 12 independently configurable channels (requests): 7 for DMA1 and 5 for DMA2
- Each of the 12 channels is connected to dedicated hardware DMA requests, software trigger is also supported on each channel. This configuration is done by software.
- Priorities between requests from channels of one DMA are software programmable (4 levels consisting of *very high*, *high*, *medium*, *low*) or hardware in case of equality (request 1 has priority over request 2, etc.)
- Independent source and destination transfer size (byte, half word, word), emulating packing and unpacking. Source/destination addresses must be aligned on the data size.
- Support for circular buffer management
- 3 event flags (DMA Half Transfer, DMA Transfer complete and DMA Transfer Error) logically ORed together in a single interrupt request for each channel
- Memory-to-memory transfer
- Peripheral-to-memory and memory-to-peripheral, and peripheral-to-peripheral transfers
- Access to Flash, SRAM, peripheral SRAM, APB1, APB2 and AHB peripherals as source and destination
- Programmable number of data to be transferred: up to 65536

The block diagram is shown in *Figure 17*.

Figure 17. DMA block diagram



1. The DMA2 controller is available only in high-density devices.
2. ADC3, SPI/I2S3, UART4, SDIO, TIM5, TIM6, DAC, TIM7, TIM8 DMA requests are available only in high-density devices.

9.3 DMA functional description

The DMA controller performs direct memory transfer by sharing the system bus with the Cortex™-M3 core. The DMA request may stop the CPU access to the system bus for some bus cycles, when the CPU and DMA are targeting the same destination (memory or peripheral). The bus matrix implements round-robin scheduling, thus ensuring at least half of the system bus bandwidth (both to memory and peripheral) for the CPU.

9.3.1 DMA transactions

After an event, the peripheral sends a request signal to the DMA Controller. The DMA controller serves the request depending on the channel priorities. As soon as the DMA Controller accesses the peripheral, an Acknowledge is sent to the peripheral by the DMA Controller. The peripheral releases its request as soon as it gets the Acknowledge from the DMA Controller. Once the request is deasserted by the peripheral, the DMA Controller release the Acknowledge. If there are more requests, the peripheral can initiate the next transaction.

In summary, each DMA transfer consists of three operations:

- A load from the peripheral data register or a location in memory addressed through the DMA_CMARx register
- A store of the data loaded to the peripheral data register or a location in memory addressed through the DMA_CMARx register
- A post-decrement of the DMA_CNDTRx register, which contains the number of transactions that have still to be performed.

9.3.2 Arbiter

The arbiter manages the channel requests based on their priority and launches the peripheral/memory access sequences.

The priorities are managed in two stages:

- Software: each channel priority can be configured in the DMA_CCRx register. There are four levels:
  - Very high priority
  - High priority
  - Medium priority
  - Low priority
- Hardware: if 2 requests have the same software priority level, the channel with the lowest number will get priority versus the channel with the highest number. For example, channel 2 gets priority over channel 4.

Note: In high-density devices, the DMA1 controller has priority over the DMA2 controller.

9.3.3 DMA channels

Each channel can handle DMA transfer between a peripheral register located at a fixed address and a memory address. The amount of data to be transferred (up to 65535) is programmable. The register which contains the amount of data items to be transferred is decremented after each transaction.

Programmable data sizes

Transfer data sizes of the peripheral and memory are fully programmable through the PSIZE and MSIZE bits in the DMA_CCRx register.

Pointer incrementation

Peripheral and memory pointers can optionally be automatically post-incremented after each transaction depending on the PINC and MINC bits in the DMA_CCRx register. If incremented mode is enabled, the address of the next transfer will be the address of the previous one incremented by 1, 2 or 4 depending on the chosen data size. The first transfer address will be the one programmed in the DMA_CPARx/DMA_CMARx registers.

If the channel is configured in non-circular mode, no DMA requests are served after the end of the transfer (i.e. once the number of data to be transferred reaches zero).

Channel configuration procedure

The following sequence should be followed to configure a DMA channelx (where x is the channel number).

1. Set the peripheral register address in the DMA_CPARx register. The data will be moved from/ to this address to/ from the memory after the peripheral event.
2. Set the memory address in the DMA_CMARx register. The data will be written to or read from this memory after the peripheral event.
3. Configure the total number of data to be transferred in the DMA_CNDTRx register. After each peripheral event, this value will be decremented.
4. Configure the channel priority using the PL[1:0] bits in the DMA_CCRx register
5. Configure data transfer direction, circular mode, peripheral & memory incremented mode, peripheral & memory data size, and interrupt after half and/or full transfer in the DMA_CCRx register
6. Activate the channel by setting the ENABLE bit in the DMA_CCRx register.

As soon as the channel is enabled, it can serve any DMA request from the peripheral connected on the channel.

Once half of the bytes are transferred, the half-transfer flag (HTIF) is set and an interrupt is generated if the Half-Transfer Interrupt Enable bit (HTIE) is set. At the end of the transfer, the Transfer Complete Flag (TCIF) is set and an interrupt is generated if the Transfer Complete Interrupt Enable bit (TCIE) is set.

Circular mode

Circular mode is available to handle circular buffers and continuous data flows (e.g. ADC scan mode). This feature can be enabled using the CIRC bit in the DMA_CCRx register. When circular mode is activated, the number of data to be transferred is automatically reloaded with the initial value programmed during the channel configuration phase, and the DMA requests continue to be served.

Memory-to-memory mode

The DMA channels can also work without being triggered by a request from a peripheral. This mode is called Memory to Memory mode.

If the MEM2MEM bit in the DMA_CCRx register is set, then the channel initiates transfers as soon as it is enabled by software by setting the Enable bit (EN) in the DMA_CCRx register. The transfer stops once the DMA_CNDTRx register reaches zero. Memory to Memory mode may not be used at the same time as Circular mode.

9.3.4 Programmable data width, data alignment and endians

When PSIZE and MSIZE are not equal, the DMA performs some data alignments as described in *Table 38: Programmable data width & endian behavior (when bits PINC = MINC = 1)*.

Table 38. Programmable data width & endian behavior (when bits PINC = MINC = 1)

Source port width	Destination port width	Number of data items to transfer (NDT)	Source content: address / data	Transfer operations	Destination content: address / data
8	8	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: READ B0[7:0] @0x0 then WRITE B0[7:0] @0x0 2: READ B1[7:0] @0x1 then WRITE B1[7:0] @0x1 3: READ B2[7:0] @0x2 then WRITE B2[7:0] @0x2 4: READ B3[7:0] @0x3 then WRITE B3[7:0] @0x3	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3
8	16	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: READ B0[7:0] @0x0 then WRITE 00B0[15:0] @0x0 2: READ B1[7:0] @0x1 then WRITE 00B1[15:0] @0x2 3: READ B3[7:0] @0x2 then WRITE 00B2[15:0] @0x4 4: READ B4[7:0] @0x3 then WRITE 00B3[15:0] @0x6	@0x0 / 00B0 @0x2 / 00B1 @0x4 / 00B2 @0x6 / 00B3
8	32	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: READ B0[7:0] @0x0 then WRITE 000000B0[31:0] @0x0 2: READ B1[7:0] @0x1 then WRITE 000000B1[31:0] @0x4 3: READ B3[7:0] @0x2 then WRITE 000000B2[31:0] @0x8 4: READ B4[7:0] @0x3 then WRITE 000000B3[31:0] @0xC	@0x0 / 000000B0 @0x4 / 000000B1 @0x8 / 000000B2 @0xC / 000000B3
16	8	4	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6	1: READ B1B0[15:0] @0x0 then WRITE B0[7:0] @0x0 2: READ B3B2[15:0] @0x2 then WRITE B2[7:0] @0x1 3: READ B5B4[15:0] @0x4 then WRITE B4[7:0] @0x2 4: READ B7B6[15:0] @0x6 then WRITE B6[7:0] @0x3	@0x0 / B0 @0x1 / B2 @0x2 / B4 @0x3 / B6
16	16	2	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6	1: READ B1B0[15:0] @0x0 then WRITE B1B0[15:0] @0x0 2: READ B3B2[15:0] @0x2 then WRITE B3B2[15:0] @0x2 3: READ B5B4[15:0] @0x4 then WRITE B5B4[15:0] @0x4 4: READ B7B6[15:0] @0x6 then WRITE B7B6[15:0] @0x6	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6
16	32	4	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6	1: READ B1B0[15:0] @0x0 then WRITE 0000B1B0[31:0] @0x0 2: READ B3B2[15:0] @0x2 then WRITE 0000B3B2[31:0] @0x4 3: READ B5B4[15:0] @0x4 then WRITE 0000B5B4[31:0] @0x8 4: READ B7B6[15:0] @0x6 then WRITE 0000B7B6[31:0] @0xC	@0x0 / 0000B1B0 @0x4 / 0000B3B2 @0x8 / 0000B5B4 @0xC / 0000B7B6
32	8	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC	1: READ B3B2B1B0[31:0] @0x0 then WRITE B0[7:0] @0x0 2: READ B7B6B5B4[31:0] @0x4 then WRITE B4[7:0] @0x1 3: READ BBBAB9B8[31:0] @0x8 then WRITE B8[7:0] @0x2 4: READ BFBEBDBC[31:0] @0xC then WRITE BC[7:0] @0x3	@0x0 / B0 @0x1 / B4 @0x2 / B8 @0x3 / BC
32	16	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC	1: READ B3B2B1B0[31:0] @0x0 then WRITE B1B0[7:0] @0x0 2: READ B7B6B5B4[31:0] @0x4 then WRITE B5B4[7:0] @0x1 3: READ BBBAB9B8[31:0] @0x8 then WRITE B9B8[7:0] @0x2 4: READ BFBEBDBC[31:0] @0xC then WRITE BDBC[7:0] @0x3	@0x0 / B1B0 @0x2 / B5B4 @0x4 / B9B8 @0x6 / BDBC
32	32	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC	1: READ B3B2B1B0[31:0] @0x0 then WRITE B3B2B1B0[31:0] @0x0 2: READ B7B6B5B4[31:0] @0x4 then WRITE B7B6B5B4[31:0] @0x4 3: READ BBBAB9B8[31:0] @0x8 then WRITE BBBAB9B8[31:0] @0x8 4: READ BFBEBDBC[31:0] @0xC then WRITE BFBEBDBC[31:0] @0xC	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC

Addressing an AHB peripheral that does not support byte or halfword write operations

When the DMA initiates an AHB byte or halfword write operation, the data are duplicated on the unused lanes of the HWDATA[31:0] bus. So when the used AHB slave peripheral does not support byte or halfword write operations (when HSIZE is not used by the peripheral) *and* does not generate any error, the DMA writes the 32 HWDATA bits as shown in the two examples below:

- To write the halfword “0xABCD”, the DMA sets the HWDATA bus to “0xABCDABCD” with HSIZE = HalfWord
- To write the byte “0xAB”, the DMA sets the HWDATA bus to “0xABABABAB” with HSIZE = Byte

Assuming that the AHB/APB bridge is an AHB 32-bit slave peripheral that does not take the HSIZE data into account, it will transform any AHB byte or halfword operation into a 32-bit APB operation in the following manner:

- an AHB byte write operation of the data “0xB0” to 0x0 (or to 0x1, 0x2 or 0x3) will be converted to an APB word write operation of the data “0xB0B0B0B0” to 0x0
- an AHB halfword write operation of the data “0xB1B0” to 0x0 (or to 0x2) will be converted to an APB word write operation of the data “0xB1B0B1B0” to 0x0

For instance, if you want to write the APB backup registers (16-bit registers aligned to a 32-bit address boundary), you must configure the memory source size (MSIZE) to “16-bit” and the peripheral destination size (PSIZE) to “32-bit”.

9.3.5 Error management

A DMA transfer error can be generated by reading from or writing to a reserved address space. When a DMA transfer error occurs during a DMA read or a write access, the faulty channel is automatically disabled through a hardware clear of its EN bit in the corresponding Channel configuration register (DMA_CCRx). The channel's transfer error interrupt flag (TEIF) in the DMA_IFR register is set and an interrupt is generated if the transfer error interrupt enable bit (TEIE) in the DMA_CCRx register is set.

9.3.6 Interrupts

An interrupt can be produced on a Half-transfer, Transfer complete or Transfer error for each DMA channel. Separate interrupt enable bits are available for flexibility.

Table 39. DMA interrupt requests

Interrupt event	Event flag	Enable Control bit
Half-transfer	HTIF	HTIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE

Note: In high-density devices, DMA2 Channel4 and DMA2 Channel5 interrupts are mapped onto the same interrupt vector. All other DMA1 and DMA2 Channel interrupts have their own interrupt vector.

9.3.7 DMA request mapping

DMA1 controller

The 7 requests from the peripherals (TIMx[1,2,3,4], ADC1, SPI1, SPI/I2S2, I2Cx[1,2] and USARTx[1,2,3]) are simply logically ORed before entering DMA1, this means that only one request must be enabled at a time. Refer to *Figure 18: DMA1 request mapping*.

The peripheral DMA requests can be independently activated/de-activated by programming the DMA control bit in the registers of the corresponding peripheral.

Figure 18. DMA1 request mapping

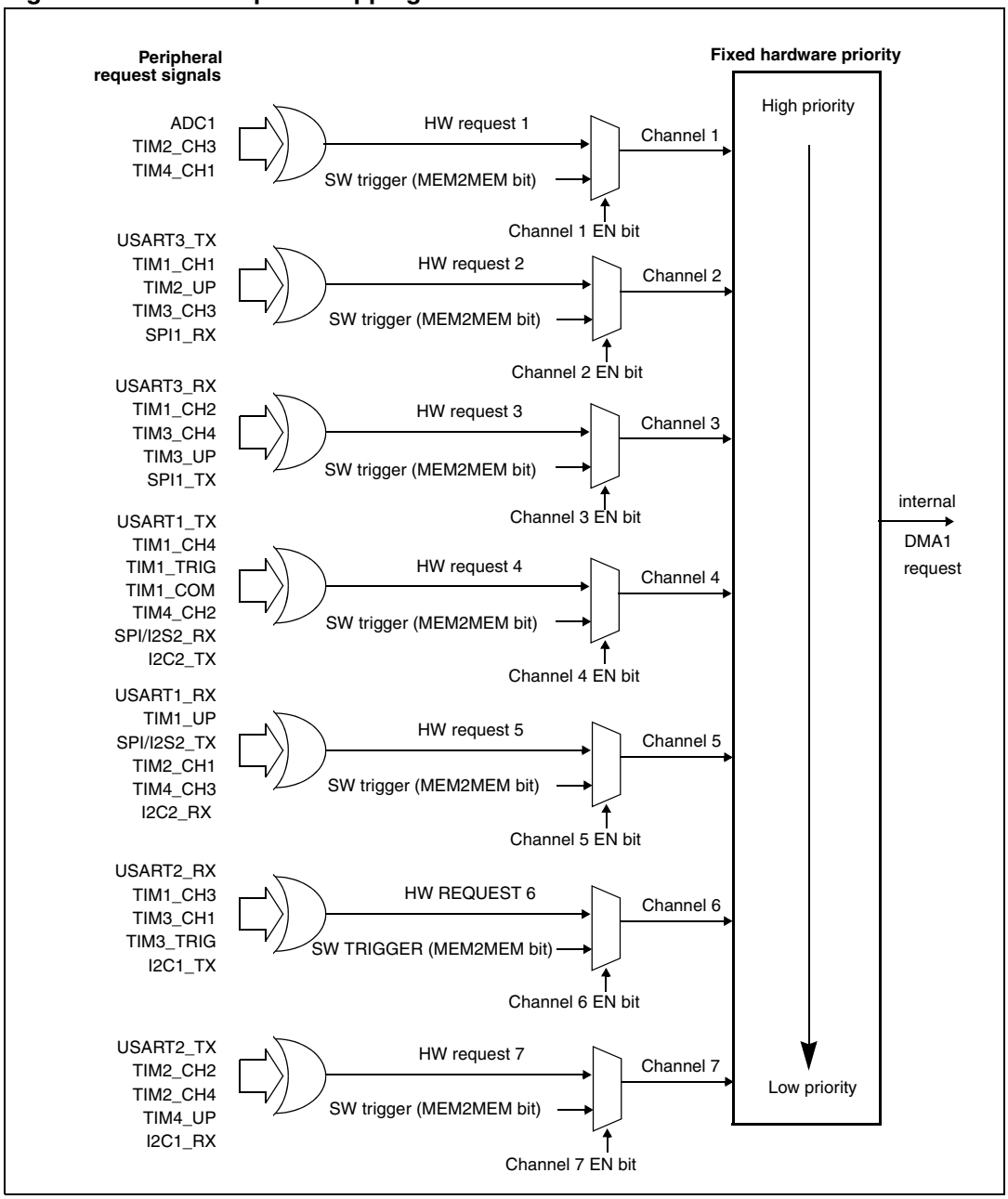


Table 40 lists the DMA requests for each channel.

Table 40. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI/I^2S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

DMA2 controller

The 5 requests from the peripherals (TIMx[5,6,7,8], ADC3, SPI/I2S3, UART4, DAC_Channel[1,2]and SDIO) are simply logically ORed before entering to the DMA2, this means that only one request must be enabled at a time. Refer to *Figure 19: DMA2 request mapping*.

The peripheral DMA requests can be independently activated/de-activated by programming the DMA control bit in the registers of the corresponding peripheral.

Note: The DMA2 controller and its relative requests are available only in high-density devices.

Figure 19. DMA2 request mapping

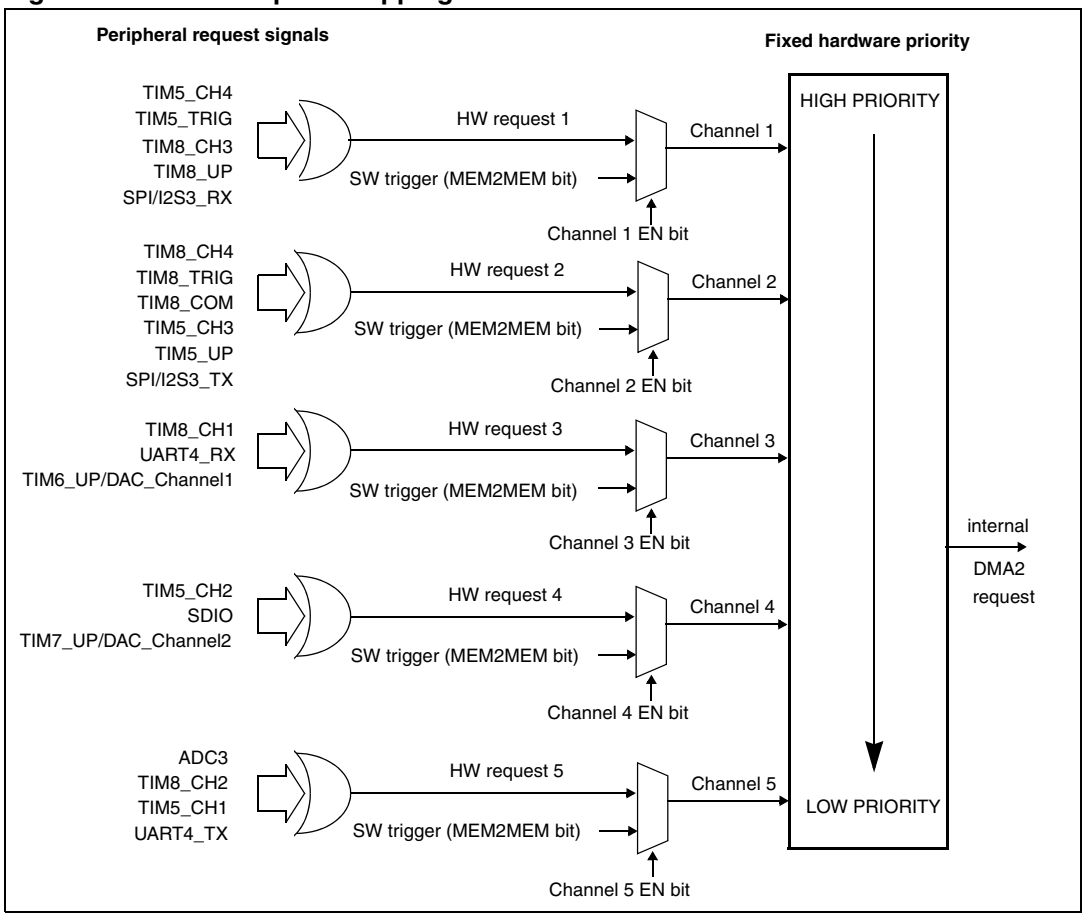


Table 41 lists the DMA2 requests for each channel.

Table 41. Summary of DMA2 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC3					ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX			
UART4			UART4_RX		UART4_TX
SDIO				SDIO	
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP		TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1			TIM6_UP/ DAC_Channel1		
TIM7/ DAC_Channel2				TIM7_UP/ DAC_Channel2	
TIM8	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

9.4 DMA registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in the register descriptions.

Note: In the following registers, all bits relative to channel6 and channel7 are not relevant for DMA2 since it has only 5 channels.

9.4.1 DMA interrupt status register (DMA_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, always read as 0.

Bits 27, 23, 19, 15, 11, 7, 3 **TEIFx:** Channel x transfer error flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No transfer error (TE) on channel x
1: A transfer error (TE) occurred on channel x

Bits 26, 22, 18, 14, 10, 6, 2 **HTIFx:** Channel x half transfer flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No half transfer (HT) event on channel x
1: A half transfer (HT) event occurred on channel x

Bits 25, 21, 17, 13, 9, 5, 1 **TCIFx:** Channel x transfer complete flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No transfer complete (TC) event on channel x
1: A transfer complete (TC) event occurred on channel x

Bits 24, 20, 16, 12, 8, 4, 0 **GIFx:** Channel x global interrupt flag (x = 1 ..7)
This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_IFCR register.
0: No TE, HT or TC event on channel x
1: A TE, HT or TC event occurred on channel x

9.4.2 DMA interrupt flag clear register (DMA_IFCR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CTEIF 7	CHTIF 7	CTCIF 7	CGIF 7	CTEIF 6	CHTIF 6	CTCIF 6	CGIF 6	CTEIF 5	CHTIF 5	CTCIF 5	CGIF 5
				w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEIF 4	CHTIF 4	CTCIF 4	CGIF 4	CTEIF 3	CHTIF 3	CTCIF 3	CGIF 3	CTEIF 2	CHTIF 2	CTCIF 2	CGIF 2	CTEIF 1	CHTIF 1	CTCIF 1	CGIF 1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:28 Reserved, always read as 0.

Bits 27, 23, 19, 15, 11, 7, 3 **CTEIFx:** *channel x transfer error clear (x = 1 ..7)*
This bit is set and cleared by software.
0: No effect
1: Clears the corresponding TEIF flag in the DMA_ISR register

Bits 26, 22, 18, 14, 10, 6, 2 **CHTIFx:** Channel x half transfer clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the corresponding HTIF flag in the DMA_ISR register

Bits 25, 21, 17, 13, 9, 5, 1 **CTCIFx:** Channel x transfer complete clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the corresponding TCIF flag in the DMA_ISR register

Bits 24, 20, 16, 12, 8, 4, 0 **CGIFx:** Channel x global interrupt clear (x = 1 ..7)
This bit is set and cleared by software.
0: No effect
1: Clears the GIF, TEIF, HTIF and TCIF flags in the DMA_ISR register

9.4.3 DMA channel x configuration register (DMA_CCRx) (x = 1 ..7)

Address offset: 0x08 + 20d × Channel number

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2 MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, always read as 0.

Bit 14 **MEM2MEM:** Memory to memory mode
This bit is set and cleared by software.
0: Memory to memory mode disabled
1: Memory to memory mode enabled

Bits 13:12 **PL[1:0]:** Channel Priority level
These bits are set and cleared by software.
00: Low
01: Medium
10: High
11: Very high

Bits 11:10 **MSIZE[1:0]:** Memory size
These bits are set and cleared by software.
00: 8-bits
01: 16-bits
10: 32-bits
11: Reserved

Bits 9:8 **PSIZE[1:0]:** Peripheral size
These bits are set and cleared by software.
00: 8-bits
01: 16-bits
10: 32-bits
11: Reserved

Bit 7 **MINC:** Memory increment mode
This bit is set and cleared by software.
0: Memory increment mode disabled
1: Memory increment mode enabled

Bit 6 **PINC:** Peripheral increment mode
This bit is set and cleared by software.
0: Peripheral increment mode disabled
1: Peripheral increment mode enabled

Bit 5 **CIRC:** Circular mode
This bit is set and cleared by software.
0: Circular mode disabled
1: Circular mode enabled

Bit 4 **DIR:** Data transfer direction
This bit is set and cleared by software.
0: Read from peripheral
1: Read from memory

Bit 3 **TEIE:** Transfer error interrupt enable
This bit is set and cleared by software.
0: TE interrupt disabled
1: TE interrupt enabled

Bit 2 **HTIE:** Half transfer interrupt enable
This bit is set and cleared by software.
0: HT interrupt disabled
1: HT interrupt enabled

Bit 1 **TCIE:** Transfer complete interrupt enable
This bit is set and cleared by software.
0: TC interrupt disabled
1: TC interrupt enabled

Bit 0 **EN:** Channel enable
This bit is set and cleared by software.
0: Channel disabled
1: Channel enabled

9.4.4 DMA channel x number of data register (DMA_CNDTRx) (x = 1 ..7)

Address offset: 0x0C + 20d × Channel number

Reset value: 0x0000 0000

Bits 31:16 Reserved, always read as 0.

Bits 15:0 **NDT[15:0]:** Number of data to transfer

Number of data to be transferred (0 up to 65535). This register can only be written when the channel is disabled. Once the channel is enabled, this register is read-only, indicating the remaining bytes to be transmitted. This register decrements after each DMA transfer.

Once the transfer is completed, this register can either stay at zero or be reloaded automatically by the value previously programmed if the channel is configured in auto-reload mode.

If this register is zero, no transaction can be served whether the channel is enabled or not.

9.4.5 DMA channel x peripheral address register (DMA_CPARx) (x = 1 ..7)

Address offset: 0x10 + dx20 × Channel number

Reset value: 0x0000 0000

Bits 31:0 **PA[31:0]:** Peripheral address

Base address of the peripheral data register from/to which the data will be read/written.

When PSIZE is 01 (16-bit), the PA[0] bit is ignored. Access is automatically aligned to a half-word address.
When PSIZE is 10 (32-bit), PA[1:0] are ignored. Access is automatically aligned to a word address.

9.4.6 DMA channel x memory address register (DMA_CMARx) (x = 1 ..7)

Address offset: 0x14 + dx20 × Channel number

Reset value: 0x0000 0000

Bits 31:0 **MA[31:0]:** Memory address

Base address of the memory area from/to which the data will be read/written.

When MSIZE is 01 (16-bit), the MA[0] bit is ignored. Access is automatically aligned to a half-word address.
When MSIZE is 10 (32-bit), MA[1:0] are ignored. Access is automatically aligned to a word address.

9.4.7 DMA register map

The following table gives the DMA register map and the reset values.

Table 42. DMA - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	**DMA_ISR**	Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5	TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x004	**DMA_IFCR**	Reserved				CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5	CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x008	**DMA_CCR1**	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00C	**DMA_CNDTR1**	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x010	**DMA_CPAR1**	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x014	**DMA_CMAR1**	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 42. DMA - register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x018	Reserved																																
0x01C	DMA_CCR2	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x020	DMA_CNDTR2	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x024	DMA_CPAR2	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x028	DMA_CMAR2	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x02C	Reserved																																
0x030	DMA_CCR3	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x034	DMA_CNDTR3	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x038	DMA_CPAR3	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x03C	DMA_CMAR3	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x040	Reserved																																
0x044	DMA_CCR4	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x048	DMA_CNDTR4	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04C	DMA_CPAR4	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x050	DMA_CMAR4	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x054	Reserved																																
0x058	DMA_CCR5	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 42. DMA - register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x05C	**DMA_CNDTR5**	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x060	**DMA_CPAR5**	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x064	**DMA_CMAR5**	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x068	Reserved																																
0x06C	**DMA_CCR6**	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x070	**DMA_CNDTR6**	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x074	**DMA_CPAR6**	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x078	**DMA_CMAR6**	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x07C	Reserved																																
0x080	**DMA_CCR7**	Reserved																	MEM2MEM	PL [1:0]		M SIZE [1:0]		PSIZ E [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x084	**DMA_CNDTR7**	Reserved																NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x088	**DMA_CPAR7**	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08C	**DMA_CMAR7**	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x090	Reserved																																

Refer to *Table 1 on page 35* for the register boundary addresses.

10 Analog-to-digital converter (ADC)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

10.1 ADC introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it measure signals from 16 external and two internal sources. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.

The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined high or low thresholds.

10.2 ADC main features

- 12-bit resolution
- Interrupt generation at End of Conversion, End of Injected conversion and Analog watchdog event
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel ‘n’
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual mode (on devices with 2 ADCs or more)
- ADC conversion time:
  - STM32F103xx performance line devices: 1 µs at 56 MHz (1.17 µs at 72 MHz)
  - STM32F101xx access line devices: 1 µs at 28 MHz (1.55 µs at 36 MHz)
  - STM32F102xx USB access line devices: 1.2 µs at 48 MHz
- ADC supply requirement: 2.4 V to 3.6 V
- ADC input range: $V_{REF-} \leq V_{IN} \leq V_{REF+}$
- DMA request generation during regular channel conversion

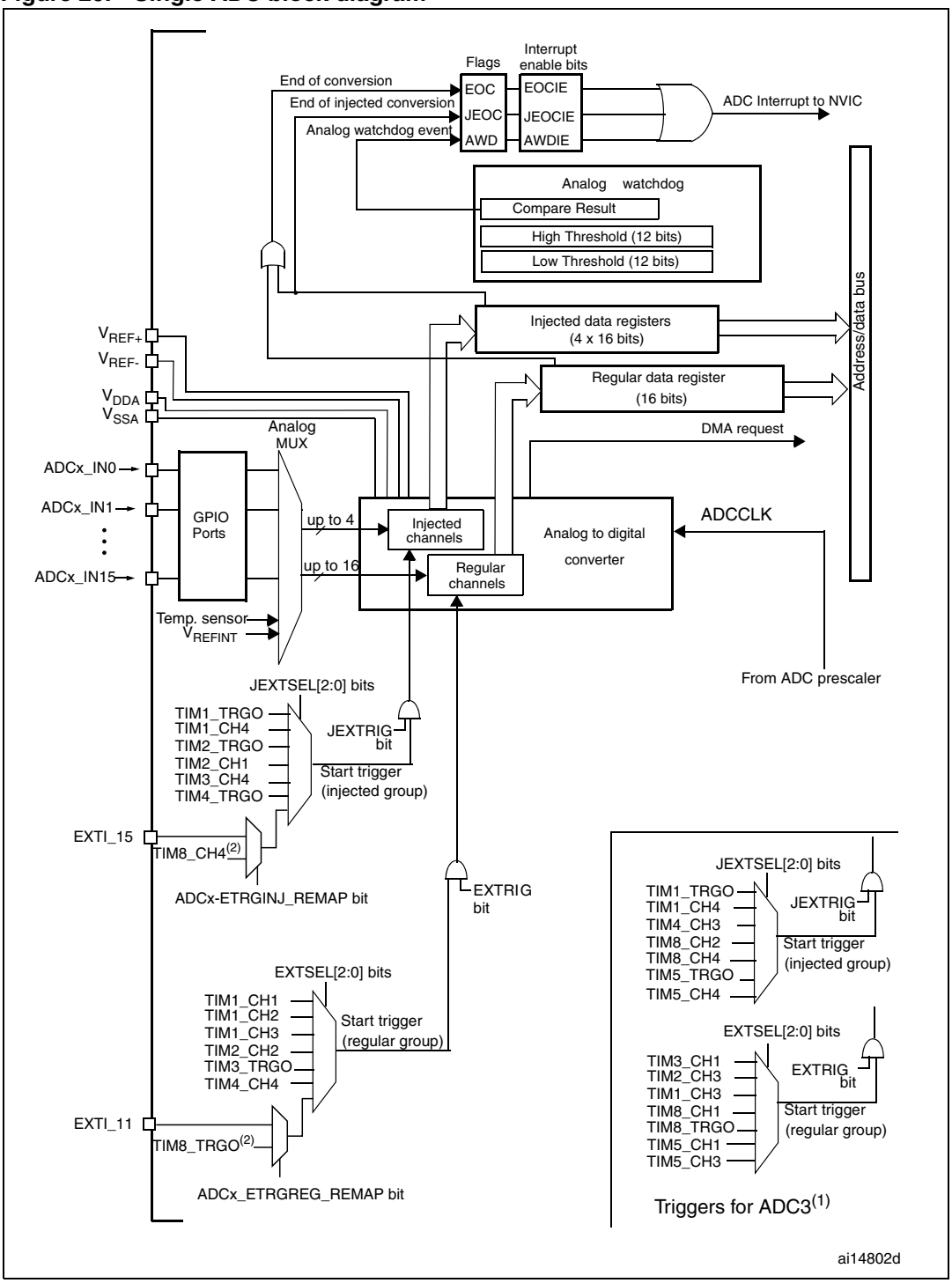
The block diagram of the ADC is shown in *Figure 20*.

Note: V_{REF-}, if available (depending on package), must be tied to V_{SSA}.

10.3 ADC functional description

Figure 20 shows a single ADC block diagrams and *Table 43* gives the ADC pin description.

Figure 20. Single ADC block diagram



1. ADC3 has regular and injected conversion triggers different from those of ADC1 and ADC2.
2. TIM8_CH4 and TIM8_TRGO with their corresponding remap bits exist only in High-density products.

Table 43. ADC pins

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, 2.4 V $\leq V_{REF+} \leq V_{DDA}$
V_{DDA}	Input, analog supply	Analog power supply equal to V_{DD} and 2.4 V $\leq V_{DDA} \leq V_{DD}$ (3.6 V)
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, $V_{REF-} = V_{SSA}$
V_{SSA}	Input, analog supply ground	Ground for analog power supply equal to V_{SS}
ADCx_IN[15:0]	Analog input signals	16 analog input channels

10.3.1 ADC on-off control

The ADC can be powered-on by setting the ADON bit in the ADC_CR2 register. When the ADON bit is set for the first time, it wakes up the ADC from Power Down mode.

Conversion starts when ADON bit is set for a second time by software after ADC power-up time (t_{STAB}).

You can stop conversion and put the ADC in power down mode by resetting the ADON bit. In this mode the ADC consumes almost no power (only a few µA).

10.3.2 ADC clock

The ADCCLK clock provided by the Clock Controller is synchronous with the PCLK2 (APB2 clock). The RCC controller has a dedicated programmable prescaler for the ADC clock, refer to *Reset and clock control (RCC) on page 66* for more details.

10.3.3 Channel selection

There are 16 multiplexed channels. It is possible to organize the conversions in two groups: regular and injected. A group consists of a sequence of conversions which can be done on any channel and in any order. For instance, it is possible to do the conversion in the following order: Ch3, Ch8, Ch2, Ch2, Ch0, Ch2, Ch2, Ch15.

- The **regular group** is composed of up to 16 conversions. The regular channels and their order in the conversion sequence must be selected in the ADC_SQRx registers. The total number of conversions in the regular group must be written in the L[3:0] bits in the ADC_SQR1 register.
- The **injected group** is composed of up to 4 conversions. The injected channels and their order in the conversion sequence must be selected in the ADC_JSQR register. The total number of conversions in the injected group must be written in the L[1:0] bits in the ADC_JSQR register.

If the ADC_SQRx or ADC_JSQR registers are modified during a conversion, the current conversion is reset and a new start pulse is sent to the ADC to convert the new chosen group.

Temperature sensor/V_{REFINT} internal channels

The Temperature sensor is connected to channel ADCx_IN16 and the internal reference voltage V_{REFINT} is connected to ADCx_IN17. These two internal channels can be selected and converted as injected or regular channels.

Note: The sensor and V_{REFINT} are only available on the master ADC1 peripheral.

10.3.4 Single conversion mode

In Single conversion mode the ADC does one conversion. This mode is started either by setting the ADON bit in the ADC_CR2 register (for a regular channel only) or by external trigger (for a regular or injected channel), while the CONT bit is 0.

Once the conversion of the selected channel is complete:

- If a regular channel was converted:
  - The converted data is stored in the 16-bit ADC_DR register
  - The EOC (End Of Conversion) flag is set
  - and an interrupt is generated if the EOCIE is set.
- If an injected channel was converted:
  - The converted data is stored in the 16-bit ADC_DRJ1 register
  - The JEOC (End Of Conversion Injected) flag is set
  - and an interrupt is generated if the JEOCIE bit is set.

The ADC is then stopped.

10.3.5 Continuous conversion mode

In continuous conversion mode ADC starts another conversion as soon as it finishes one. This mode is started either by external trigger or by setting the ADON bit in the ADC_CR2 register, while the CONT bit is 1.

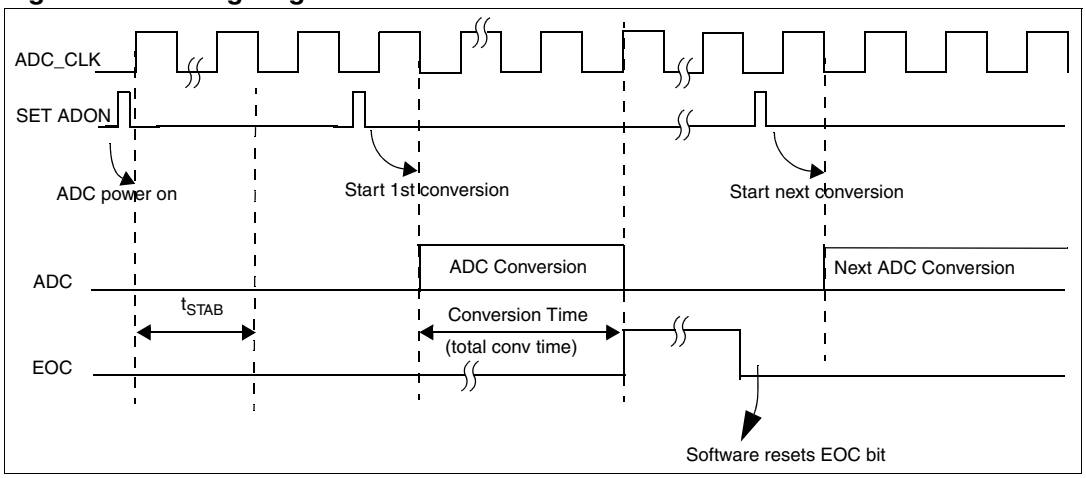
After each conversion:

- If a regular channel was converted:
  - The converted data is stored in the 16-bit ADC_DR register
  - The EOC (End Of Conversion) flag is set
  - An interrupt is generated if the EOCIE is set.
- If an injected channel was converted:
  - The converted data is stored in the 16-bit ADC_DRJ1 register
  - The JEOC (End Of Conversion Injected) flag is set
  - An interrupt is generated if the JEOCIE bit is set.

10.3.6 Timing diagram

As shown in *Figure 21*, the ADC needs a stabilization time of t_{STAB} before it starts converting accurately. After the start of ADC conversion and after 14 clock cycles, the EOC flag is set and the 16-bit ADC Data register contains the result of the conversion.

Figure 21. Timing diagram



10.3.7 Analog watchdog

The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a low threshold or above a high threshold. These thresholds are programmed in the 12 least significant bits of the ADC_HTR and ADC_LTR 16-bit registers. An interrupt can be enabled by using the AWDIE bit in the ADC_CR1 register.

The threshold value is independent of the alignment selected by the ALIGN bit in the ADC_CR2 register. The comparison is done before the alignment (see *Section 10.5*).

The analog watchdog can be enabled on one or more channels by configuring the ADC_CR1 register as shown in *Table 44.*

Figure 22. Analog watchdog guarded area

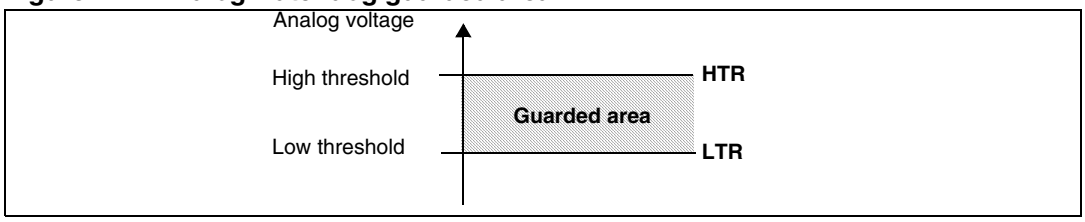


Table 44. Analog watchdog channel selection

Channels to be guarded by analog watchdog	ADC_CR1 register control bits (x = don't care)		
	AWDSGL bit	AWDEN bit	JAWDEN bit
None	x	0	0
All injected channels	0	0	1
All regular channels	0	1	0
All regular and injected channels	0	1	1
Single(1) injected channel	1	0	1
Single(1) regular channel	1	1	0
Single (1) regular or injected channel	1	1	1

1. Selected by AWDCH[4:0] bits

10.3.8 Scan mode

This mode is used to scan a group of analog channels.

Scan mode can be selected by setting the SCAN bit in the ADC_CR1 register. Once this bit is set, ADC scans all the channels selected in the ADC_SQRx registers (for regular channels) or in the ADC_JSQR (for injected channels). A single conversion is performed for each channel of the group. After each end of conversion the next channel of the group is converted automatically. If the CONT bit is set, conversion does not stop at the last selected group channel but continues again from the first selected group channel.

If the DMA bit is set, the direct memory access controller is used to transfer the converted data of regular group channels to SRAM after each EOC.

The injected channel converted data is always stored in the ADC_JDRx registers.

10.3.9 Injected channel management

Triggered injection

To use triggered injection, the JAUTO bit must be cleared and SCAN bit must be set in the ADC_CR1 register.

1. Start conversion of a group of regular channels either by external trigger or by setting the ADON bit in the ADC_CR2 register.
2. If an external injected trigger occurs during the regular group channel conversion, the current conversion is reset and the injected channel sequence is converted in Scan once mode.
3. Then, the regular group channel conversion is resumed from the last interrupted regular conversion. If a regular event occurs during an injected conversion, it doesn't interrupt it but the regular sequence is executed at the end of the injected sequence. *Figure 23* shows the timing diagram.

Note: When using triggered injection, one must ensure that the interval between trigger events is longer than the injection sequence. For instance, if the sequence length is 28 ADC clock cycles (that is two conversions with a 1.5 clock-period sampling time), the minimum interval between triggers must be 29 ADC clock cycles.

Auto-injection

If the JAUTO bit is set, then the injected group channels are automatically converted after the regular group channels. This can be used to convert a sequence of up to 20 conversions programmed in the ADC_SQRx and ADC_JSQR registers.

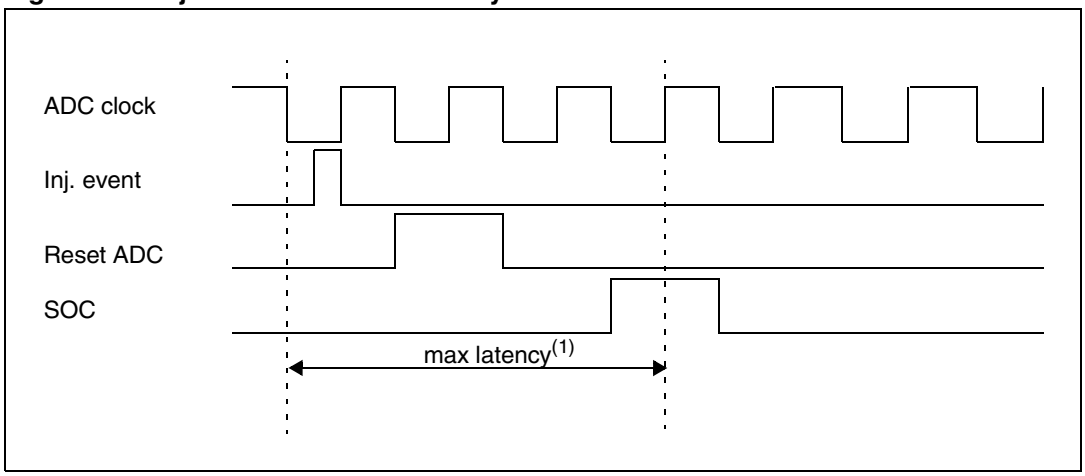
In this mode, external trigger on injected channels must be disabled.

If the CONT bit is also set in addition to the JAUTO bit, regular channels followed by injected channels are continuously converted.

For ADC clock prescalers ranging from 4 to 8, a delay of 1 ADC clock period is automatically inserted when switching from regular to injected sequence (respectively injected to regular). When the ADC clock prescaler is set to 2, the delay is 2 ADC clock periods.

Note: It is not possible to use both auto-injected and discontinuous modes simultaneously.

Figure 23. Injected conversion latency



1. The maximum latency value can be found in the electrical characteristics of the STM32F101xx and STM32F103xx datasheets.

10.3.10 Discontinuous mode

Regular group

This mode is enabled by setting the DISCEN bit in the ADC_CR1 register. It can be used to convert a short sequence of n conversions (n <=8) which is a part of the sequence of conversions selected in the ADC_SQRx registers. The value of n is specified by writing to the DISCNUM[2:0] bits in the ADC_CR1 register.

When an external trigger occurs, it starts the next n conversions selected in the ADC_SQRx registers until all the conversions in the sequence are done. The total sequence length is defined by the L[3:0] bits in the ADC_SQR1 register.

Example:

n = 3, channels to be converted = 0, 1, 2, 3, 6, 7, 9, 10
1st trigger: sequence converted 0, 1, 2
2nd trigger: sequence converted 3, 6, 7
3rd trigger: sequence converted 9, 10 and an EOC event generated
4th trigger: sequence converted 0, 1, 2

Note: When a regular group is converted in discontinuous mode, no rollover will occur.

When all sub groups are converted, the next trigger starts conversion of the first sub-group. In the example above, the 4th trigger reconverts the 1st sub-group channels 0, 1 and 2.

Injected group

This mode is enabled by setting the JDISCEN bit in the ADC_CR1 register. It can be used to convert the sequence selected in the ADC_JSQR register, channel by channel, after an external trigger event.

When an external trigger occurs, it starts the next channel conversions selected in the ADC_JSQR registers until all the conversions in the sequence are done. The total sequence length is defined by the JL[1:0] bits in the ADC_JSQR register.

Example:

n = 1, channels to be converted = 1, 2, 3
1st trigger: channel 1 converted
2nd trigger: channel 2 converted
3rd trigger: channel 3 converted and EOC and JEOC events generated
4th trigger: channel 1

Note: 1 When all injected channels are converted, the next trigger starts the conversion of the first injected channel. In the example above, the 4th trigger reconverts the 1st injected channel 1.

2 It is not possible to use both auto-injected and discontinuous modes simultaneously.

3 The user must avoid setting discontinuous mode for both regular and injected groups together. Discontinuous mode must be enabled only for one group conversion.

10.4 Calibration

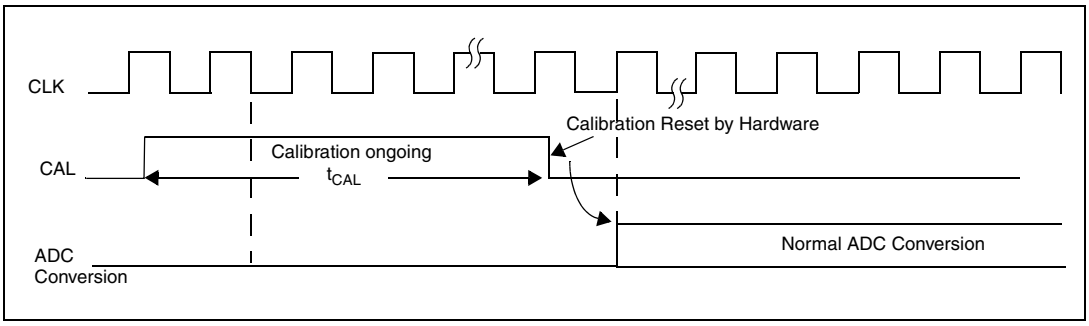
The ADC has an built-in self calibration mode. Calibration significantly reduces accuracy errors due to internal capacitor bank variations. During calibration, an error-correction code (digital word) is calculated for each capacitor, and during all subsequent conversions, the error contribution of each capacitor is removed using this code.

Calibration is started by setting the CAL bit in the ADC_CR2 register. Once calibration is over, the CAL bit is reset by hardware and normal conversion can be performed. It is recommended to calibrate the ADC once at power-on. The calibration codes are stored in the ADC_DR as soon as the calibration phase ends.

Note: 1 It is recommended to perform a calibration after each power-up.

2 Before starting a calibration the ADC must have been in power-off state (ADON bit = '0') for at least two ADC clock cycles.

Figure 24. Calibration timing diagram



10.5 Data alignment

ALIGN bit in the ADC_CR2 register selects the alignment of data stored after conversion. Data can be left or right aligned as shown in *Figure 25.* and *Figure 26.*

The injected group channels converted data value is decreased by the user-defined offset written in the ADC_JOFRx registers so the result can be a negative value. The SEXT bit is the extended sign value.

For regular group channels no offset is subtracted so only twelve bits are significant.

Figure 25. Right alignment of data

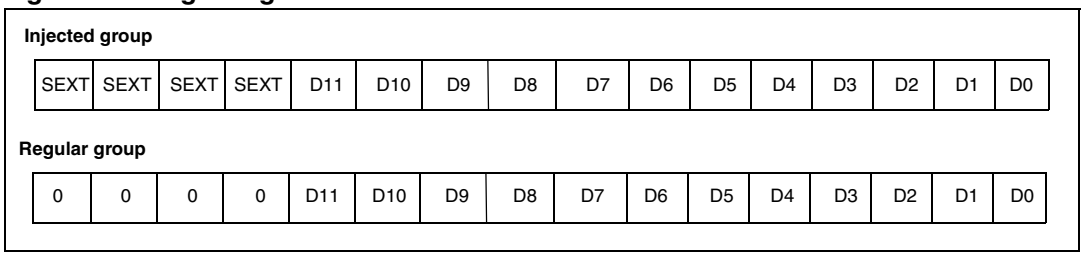
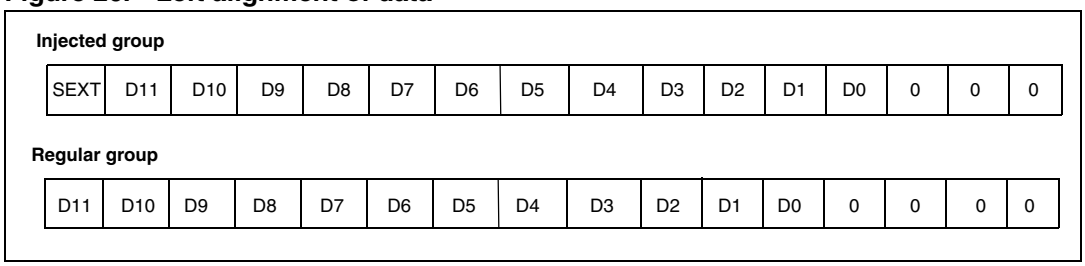


Figure 26. Left alignment of data



10.6 Channel-by-channel programmable sample time

ADC samples the input voltage for a number of ADC_CLK cycles which can be modified using the SMP[2:0] bits in the ADC_SMPR1 and ADC_SMPR2 registers. Each channel can be sampled with a different sample time.

The total conversion time is calculated as follows:

Tconv = Sampling time + 12.5 cycles

Example:

With an ADCCLK = 14 MHz and a sampling time of 1.5 cycles:

Tconv = 1.5 + 12.5 = 14 cycles = 1µs

10.7 Conversion on external trigger

Conversion can be triggered by an external event (e.g. timer capture, EXTI line). If the EXTTRIG control bit is set then external events are able to trigger a conversion. The EXTSEL[2:0] and JEXTSEL[2:0] control bits allow the application to select decide which out of 8 possible events can trigger conversion for the regular and injected groups.

Note: When an external trigger is selected for ADC regular or injected conversion, only the rising edge of the signal can start the conversion.

Table 45. External trigger for regular channels for ADC1 and ADC2

Source	Type	EXTSEL[2:0]
TIM1_CC1 event	Internal signal from on-chip timers	000
TIM1_CC2 event		001
TIM1_CC3 event		010
TIM2_CC2 event		011
TIM3_TRGO event		100
TIM4_CC4 event		101
EXTI line11/TIM8_TRGO event(1)(2)	External pin/Internal signal from on-chip timers	110
SWSTART	Software control bit	111

1. The TIM8_TRGO event exists only in High-density devices.
2. The selection of the external trigger EXTI line11 or TIM8_TRGO event for regular channels is done through configuration bits ADC1_ETRGREG_REMAP and ADC2_ETRGREG_REMAP for ADC1 and ADC2, respectively.

Table 46. External trigger for injected channels for ADC1 and ADC2

Source	Connection type	JEXTSEL[2:0]
TIM1_TRGO event	Internal signal from on-chip timers	000
TIM1_CC4 event		001
TIM2_TRGO event		010
TIM2_CC1 event		011
TIM3_CC4 event		100
TIM4_TRGO event		101
EXTI line15/TIM8_CC4 event(1)(2)	External pin/Internal signal from on-chip timers	110
JSWSTART	Software control bit	111

1. The TIM8_CC4 event exists only in High-density devices.
2. The selection of the external trigger EXTI line15 or TIM8_CC4 event for injected channels is done through configuration bits ADC1_ETRGINJ_REMAP and ADC2_ETRGINJ_REMAP for ADC1 and ADC2, respectively.

Table 47. External trigger for regular channels for ADC3

Source	Connection type	EXTSEL[2:0]
TIM3_CC1 event	Internal signal from on-chip timers	000
TIM2_CC3 event		001
TIM1_CC3 event		010
TIM8_CC1 event		011
TIM8_TRGO event		100
TIM5_CC1 event		101
TIM5_CC3 event		110
SWSTART	Software control bit	111

Table 48. External trigger for injected channels for ADC3

Source	Connection type	JEXTSEL[2:0]
TIM1_TRGO event	Internal signal from on-chip timers	000
TIM1_CC4 event		001
TIM4_CC3 event		010
TIM8_CC2 event		011
TIM8_CC4 event		100
TIM5_TRGO event		101
TIM5_CC4 event		110
JSWSTART	Software control bit	111

The software source trigger events can be generated by setting a bit in a register (SWSTART and JSWSTART in ADC_CR2).

A regular group conversion can be interrupted by an injected trigger.

10.8 DMA request

Since converted regular channels value are stored in a unique data register, it is necessary to use DMA for conversion of more than one regular channel. This avoids the loss of data already stored in the ADC_DR register.

Only the end of conversion of a regular channel generates a DMA request, which allows the transfer of its converted data from the ADC_DR register to the destination location selected by the user.

Note: Only ADC1 and ADC3 have this DMA capability. ADC2-converted data can be transferred in dual ADC mode using DMA thanks to master ADC1.

10.9 Dual ADC mode

In devices with two ADCs or more, dual ADC mode can be used (see *Figure 27*).

In dual ADC mode the start of conversion is triggered alternately or simultaneously by the ADC1 master to the ADC2 slave, depending on the mode selected by the DUALMOD[2:0] bits in the ADC1_CR1 register.

Note: In dual mode, when configuring conversion to be triggered by an external event, the user must set the trigger for the master only and set a software trigger for the slave to prevent spurious triggers to start unwanted slave conversion. However, external triggers must be enabled on both master and slave ADCs.

The following six possible modes are implemented:

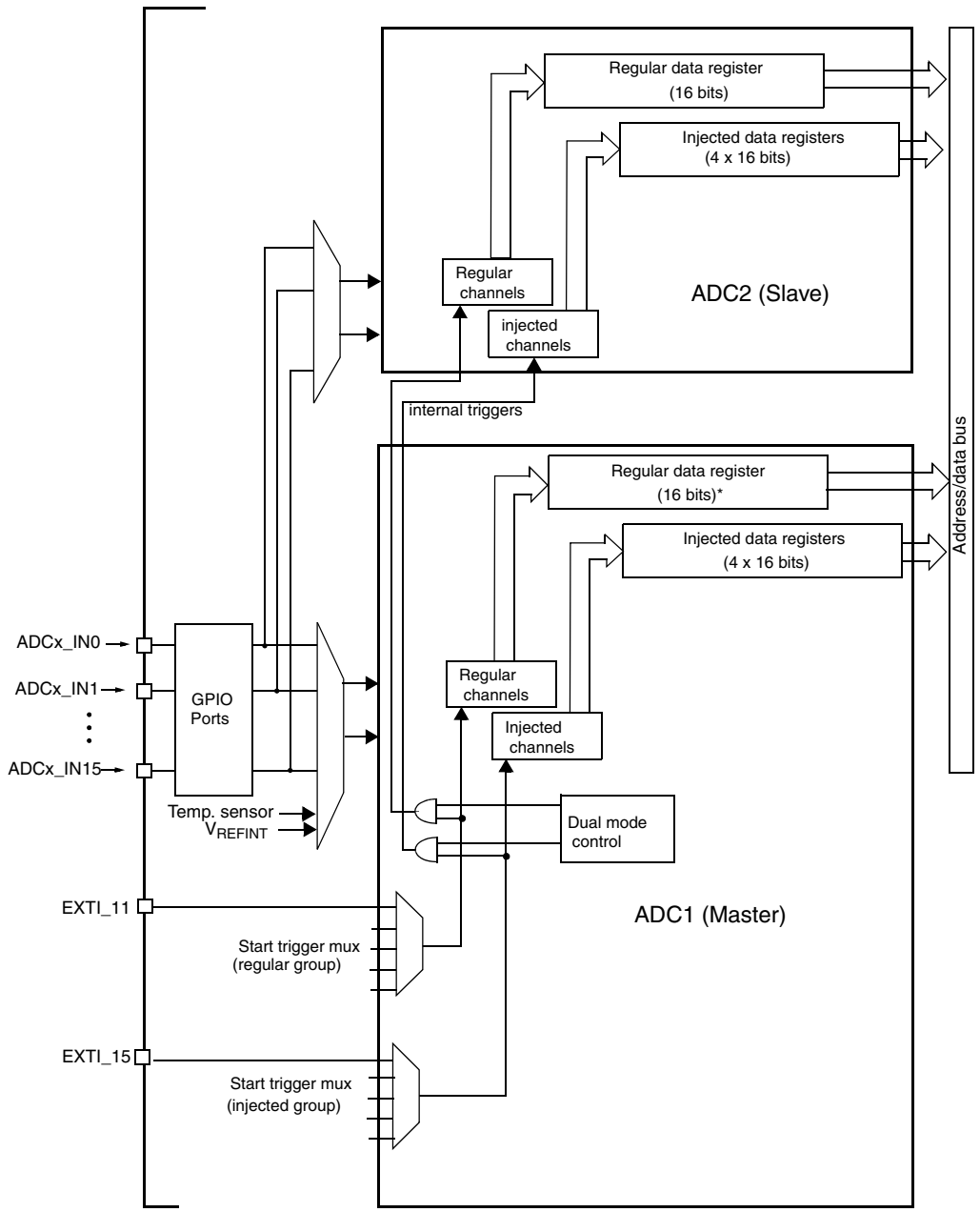
- Injected simultaneous mode
- Regular simultaneous mode
- Fast interleaved mode
- Slow interleaved mode
- Alternate trigger mode
- Independent mode

It is also possible to use the previous modes combined in the following ways:

- Injected simultaneous mode + Regular simultaneous mode
- Regular simultaneous mode + Alternate trigger mode
- Injected simultaneous mode + Interleaved mode

Note: In dual ADC mode, to read the slave converted data on the master data register, the DMA bit must be enabled even if it is not used to transfer converted regular channel data.

Figure 27. Dual ADC block diagram



Note: External triggers are present on ADC2 but are not shown for the purposes of this diagram

* In some dual ADC modes, the ADC1 data register (ADC1_DR) contains both ADC1 and ADC2 regular converted data over the entire 32 bits.

10.9.1 Injected simultaneous mode

This mode converts an injected channel group. The source of external trigger comes from the injected group mux of ADC1 (selected by the JEXTSEL[2:0] bits in the ADC1_CR2 register). A simultaneous trigger is provided to ADC2.

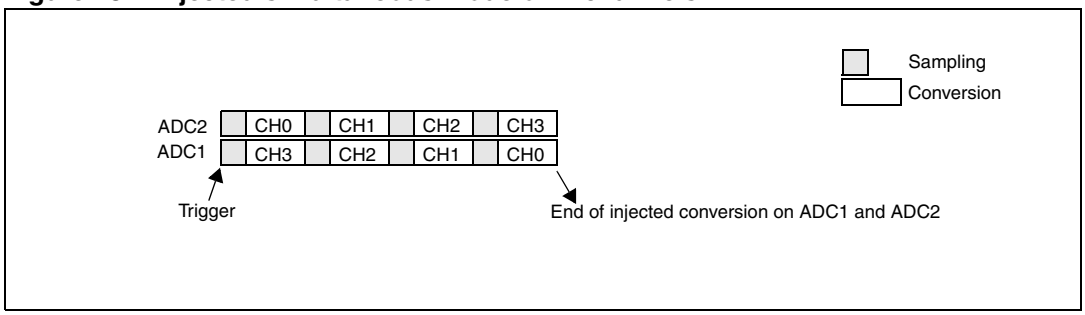
Note: Do not convert the same channel on the two ADCs (no overlapping sampling times for the two ADCs when converting the same channel).

At the end of conversion event on ADC1 or ADC2:

- The converted data is stored in the ADC_JDRx registers of each ADC interface.
- An JEOC interrupt is generated (if enabled on one of the two ADC interfaces) when the ADC1/ADC2 injected channels are all converted.

Note: In simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

Figure 28. Injected simultaneous mode on 4 channels



10.9.2 Regular simultaneous mode

This mode is performed on a regular channel group. The source of the external trigger comes from the regular group mux of ADC1 (selected by the EXTSEL[2:0] bits in the ADC1_CR2 register). A simultaneous trigger is provided to the ADC2.

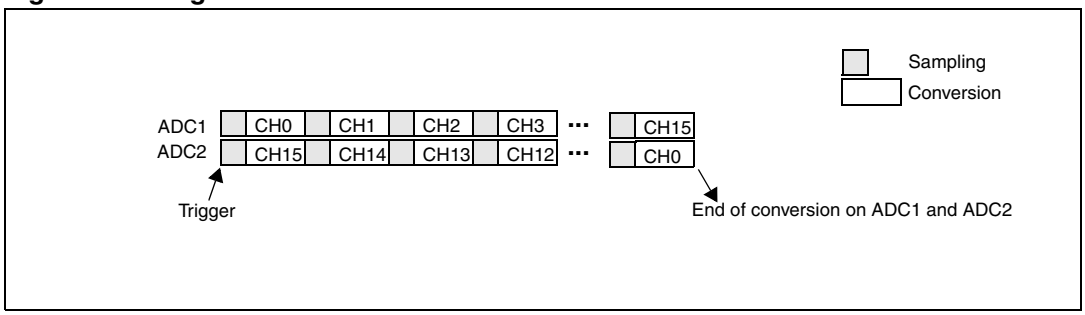
Note: Do not convert the same channel on the two ADCs (no overlapping sampling times for the two ADCs when converting the same channel).

At the end of conversion event on ADC1 or ADC2:

- A 32-bit DMA transfer request is generated (if DMA bit is set) which transfers to SRAM the ADC1_DR 32-bit register containing the ADC2 converted data in the upper halfword and the ADC1 converted data in the lower halfword.
- An EOC interrupt is generated (if enabled on one of the two ADC interfaces) when ADC1/ADC2 regular channels are all converted.

Note: In regular simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

Figure 29. Regular simultaneous mode on 16 channels



10.9.3 Fast interleaved mode

This mode can be started only on a regular channel group (usually one channel). The source of external trigger comes from the regular channel mux of ADC1. After an external trigger occurs:

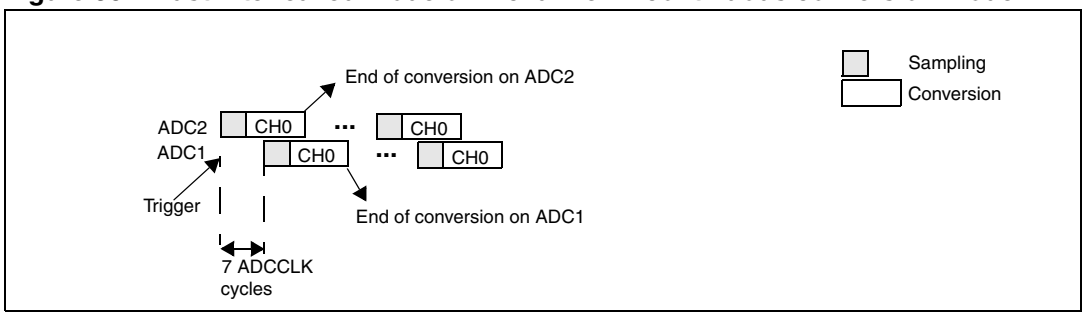
- ADC2 starts immediately and
- ADC1 starts after a delay of 7 ADC clock cycles.

If CONT bit is set on both ADC1 and ADC2 the selected regular channels of both ADCs are continuously converted.

After an EOC interrupt is generated by ADC1 (if enabled through the EOCIE bit) a 32-bit DMA transfer request is generated (if the DMA bit is set) which transfers to SRAM the ADC1_DR 32-bit register containing the ADC2 converted data in the upper halfword and the ADC1 converted data in the lower halfword.

Note: The maximum sampling time allowed is <7 ADCCLK cycles to avoid the overlap between ADC1 and ADC2 sampling phases in the event that they convert the same channel.

Figure 30. Fast interleaved mode on 1 channel in continuous conversion mode



10.9.4 Slow interleaved mode

This mode can be started only on a regular channel group (only one channel). The source of external trigger comes from regular channel mux of ADC1. After external trigger occurs:

- ADC2 starts immediately and
- ADC1 starts after a delay of 14 ADC clock cycles.
- ADC2 starts after a second delay of 14 ADC cycles, and so on.

Note: The maximum sampling time allowed is <14 ADCCLK cycles to avoid an overlap with the next conversion.

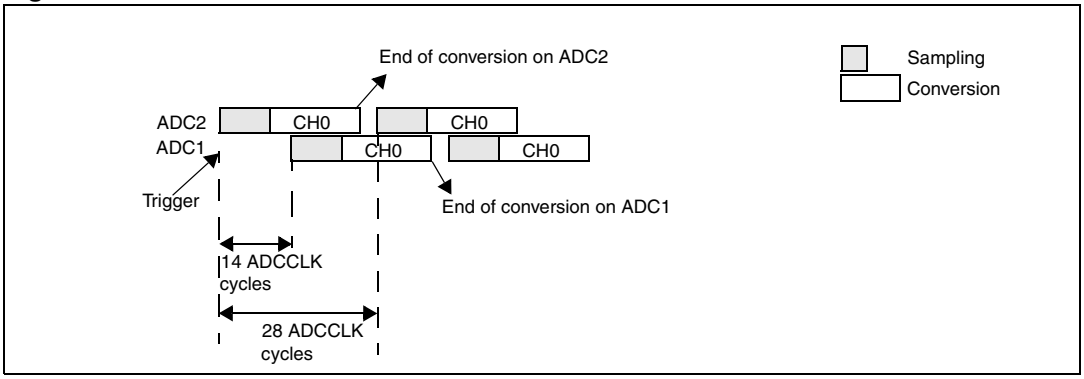
After an EOC interrupt is generated by ADC1 (if enabled through the EOCIE bit) a 32-bit DMA transfer request is generated (if the DMA bit is set) which transfers to SRAM the ADC1_DR 32-bit register containing the ADC2 converted data in the upper halfword and the ADC1 converted data in the lower halfword.

A new ADC2 start is automatically generated after 28 ADC clock cycles

CONT bit can not be set in the mode since it continuously converts the selected regular channel.

Note: The application must ensure that no external trigger for injected channel occurs when interleaved mode is enabled.

Figure 31. Slow interleaved mode on 1 channel



10.9.5 Alternate trigger mode

This mode can be started only on an injected channel group. The source of external trigger comes from the injected group mux of ADC1.

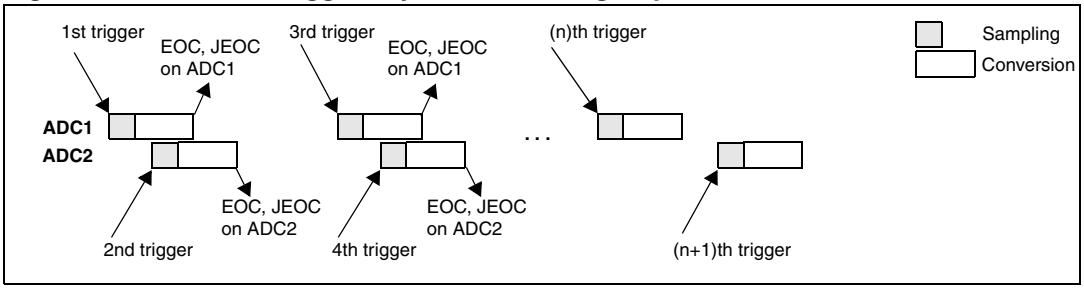
- When the 1st trigger occurs, all injected group channels in ADC1 are converted.
- When the 2nd trigger arrives, all injected group channels in ADC2 are converted
- and so on.

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC1 are converted.

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC2 are converted.

If another external trigger occurs after all injected group channels have been converted then the alternate trigger process restarts by converting ADC1 injected group channels.

Figure 32. Alternate trigger: injected channel group of each ADC



If the injected discontinuous mode is enabled for both ADC1 and ADC2:

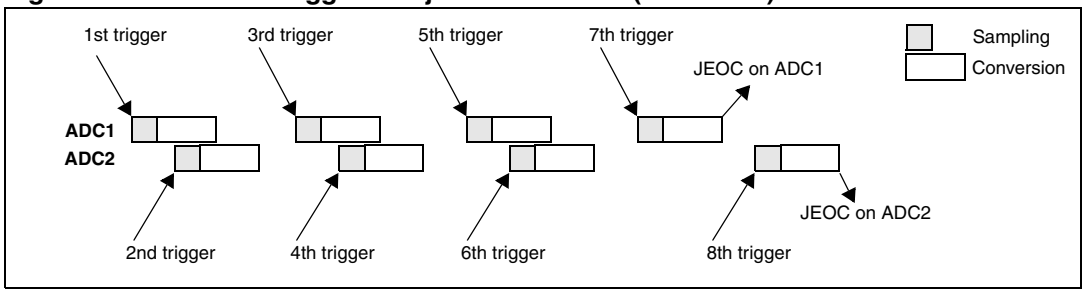
- When the 1st trigger occurs, the first injected channel in ADC1 is converted.
- When the 2nd trigger arrives, the first injected channel in ADC2 are converted
- and so on....

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC1 are converted.

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC2 are converted.

If another external trigger occurs after all injected group channels have been converted then the alternate trigger process restarts.

Figure 33. Alternate trigger: 4 injected channels (each ADC) in discontinuous model



10.9.6 Independent mode

In this mode the dual ADC synchronization is bypassed and each ADC interfaces works independently.

10.9.7 Combined regular/injected simultaneous mode

It is possible to interrupt simultaneous conversion of a regular group to start simultaneous conversion of an injected group.

Note: In combined regular/injected simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

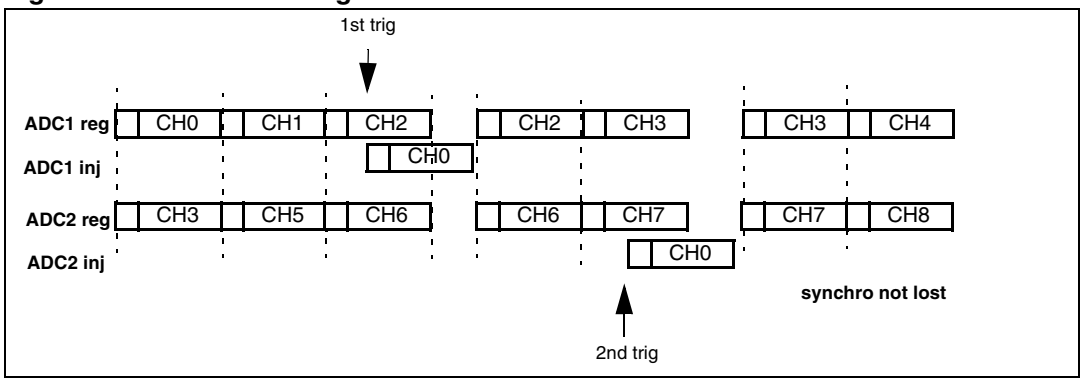
10.9.8 Combined regular simultaneous + alternate trigger mode

It is possible to interrupt regular group simultaneous conversion to start alternate trigger conversion of an injected group. *Figure 34* shows the behavior of an alternate trigger interrupting a regular simultaneous conversion.

The injected alternate conversion is immediately started after the injected event arrives. If regular conversion is already running, in order to ensure synchronization after the injected conversion, the regular conversion of both (master/slave) ADCs is stopped and resumed synchronously at the end of the injected conversion.

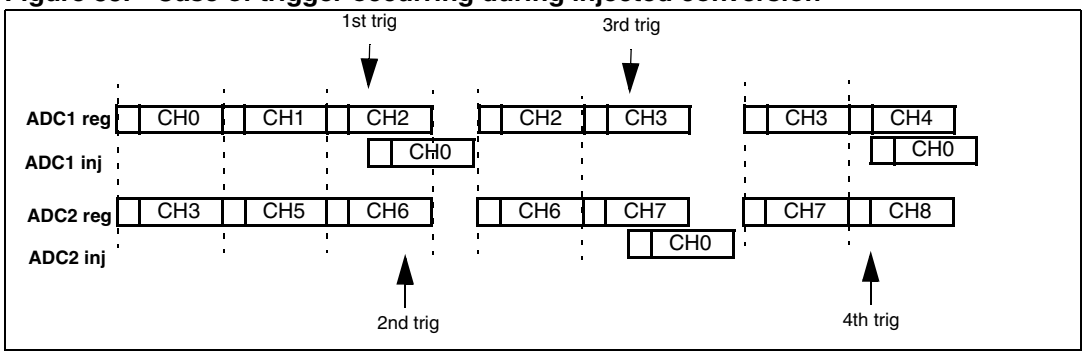
Note: In combined regular simultaneous + alternate trigger mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

Figure 34. Alternate + Regular simultaneous



If a trigger occurs during an injected conversion that has interrupted a regular conversion, it will be ignored. *Figure 35* shows the behavior in this case (2nd trig is ignored).

Figure 35. Case of trigger occurring during injected conversion

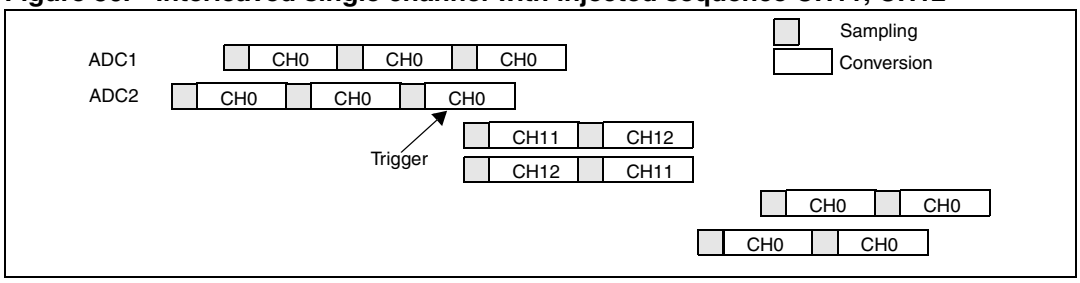


10.9.9 Combined injected simultaneous + interleaved

It is possible to interrupt an interleaved conversion with an injected event. In this case the interleaved conversion is interrupted and the injected conversion starts, at the end of the injected sequence the interleaved conversion is resumed. *Figure 36* shows the behavior using an example.

Note: When the ADC clock prescaler is set to 4, the interleaved mode does not recover with evenly spaced sampling periods: the sampling interval is 8 ADC clock periods followed by 6 ADC clock periods, instead of 7 clock periods followed by 7 clock periods.

Figure 36. Interleaved single channel with injected sequence CH11, CH12



10.10 Temperature sensor

The temperature sensor can be used to measure the ambient temperature (T_A) of the device.

The temperature sensor is internally connected to the ADCx_IN16 input channel which is used to convert the sensor output voltage into a digital value. The recommended sampling time for the temperature sensor is 17.1 µs.

The block diagram of the temperature sensor is shown in *Figure 37*.

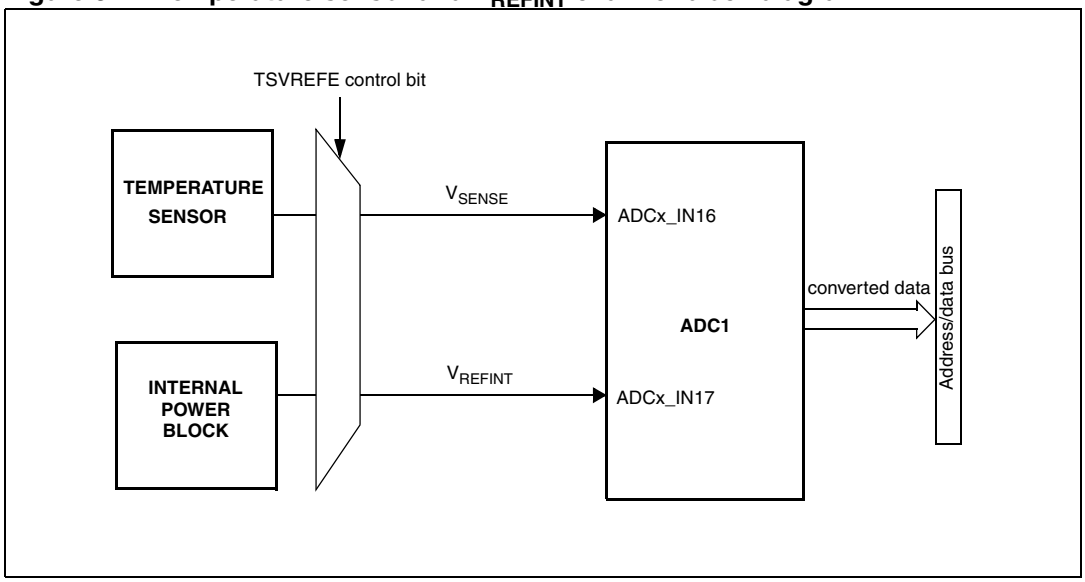
When not in use, this sensor can be put in power down mode.

Note: The TSVREFE bit must be set to enable both internal channels: ADCx_IN16 (temperature sensor) and ADCx_IN17 (V_{REFINT}) conversion.

Main features

- Supported Temperature Range: –40 to 125 degrees
- Precision: ± 1.5 °C

Figure 37. Temperature sensor and V_{REFINT} channel block diagram



Reading the temperature

To use the sensor:

1. Select the ADCx_IN16 input channel.
2. Select a sample time of 17.1 µs
3. Set the TSVREFE bit in the *ADC control register 2 (ADC_CR2)* to wake up the temperature sensor from power down mode.
4. Start the ADC conversion by setting the ADON bit (or by external trigger).
5. Read the resulting V_{SENSE} data in the ADC data register
6. Obtain the temperature using the following formula:

   Temperature (in °C) = $\{(V_{25} - V_{SENSE}) / Avg_Slope\} + 25$.

   Where,

   $V_{25} = V_{SENSE}$ value for 25° C and

   Avg_Slope = Average Slope for curve between Temperature vs. V_{SENSE} (given in mV/° C or µV/ °C).

   Refer to the Electrical characteristics section for the actual values of V_{25} and Avg_Slope.

Note: The sensor has a startup time after waking from power down mode before it can output V_{SENSE} at the correct level. The ADC also has a startup time after power-on, so to minimize the delay, the ADON and TSVREFE bits should be set at the same time.

10.11 ADC interrupts

An interrupt can be produced on end of conversion for regular and injected groups and when the analog watchdog status bit is set. Separate interrupt enable bits are available for flexibility.

Note: ADC1 and ADC2 interrupts are mapped onto the same interrupt vector. ADC3 interrupts are mapped onto a separate interrupt vector.

Two other flags are present in the ADC_SR register, but there is no interrupt associated with them:

- JSTRT (Start of conversion for injected group channels)
- STRT (Start of conversion for regular group channels)

Table 49. ADC interrupts

Interrupt event	Event flag	Enable Control bit
End of Conversion regular group	EOC	EOCIE
End of Conversion injected group	JEOC	JEOCIE
Analog watchdog status bit is set	AWD	AWDIE

10.12 ADC registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

10.12.1 ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											STRT	JSTRT	JEOC	EOC	AWD
Res.											rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bits 31:5 Reserved, must be kept cleared.

Bit 4 **STRT:** Regular channel Start flag

This bit is set by hardware when regular channel conversion starts. It is cleared by software.
0: No regular channel conversion started
1: Regular channel conversion has started

Bit 3 **JSTRT:** Injected channel Start flag

This bit is set by hardware when injected channel group conversion starts. It is cleared by software.
0: No injected group conversion started
1: Injected group conversion has started

Bit 2 **JEOC:** Injected channel end of conversion

This bit is set by hardware at the end of all injected group channel conversion. It is cleared by software.
0: Conversion is not complete
1: Conversion complete

Bit 1 **EOC:** End of conversion

This bit is set by hardware at the end of a group channel conversion (regular or injected). It is cleared by software or by reading the ADC_DR.
0: Conversion is not complete
1: Conversion complete

Bit 0 **AWD:** Analog watchdog flag

This bit is set by hardware when the converted voltage crosses the values programmed in the ADC_LTR and ADC_HTR registers. It is cleared by software.
0: No Analog watchdog event occurred
1: Analog watchdog event occurred

10.12.2 ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								AWDEN	JAWDEN	Reserved		DUALMOD[3:0]			
Res.								rw	rw	Res.		rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bit 23 **AWDEN:** Analog watchdog enable on regular channels
This bit is set/reset by software.
0: Analog watchdog disabled on regular channels
1: Analog watchdog enabled on regular channels

Bit 22 **JAWDEN:** Analog watchdog enable on injected channels
This bit is set/reset by software.
0: Analog watchdog disabled on injected channels
1: Analog watchdog enabled on injected channels

Bits 21:20 Reserved, must be kept cleared.

Bits 19:16 **DUALMOD[3:0]**: Dual mode selection
These bits are written by software to select the operating mode.
0000: Independent mode.
0001: Combined regular simultaneous + injected simultaneous mode
0010: Combined regular simultaneous + alternate trigger mode
0011: Combined injected simultaneous + fast interleaved mode
0100: Combined injected simultaneous + slow Interleaved mode
0101: Injected simultaneous mode only
0110: Regular simultaneous mode only
0111: Fast interleaved mode only
1000: Slow interleaved mode only
1001: Alternate trigger mode only

Note: These bits are reserved in ADC2 and ADC3.
In dual mode, a change of channel configuration generates a restart that can produce a loss of synchronization. It is recommended to disable dual mode before any configuration change.

Bits 15:13 **DISCNUM[2:0]**: Discontinuous mode channel count
These bits are written by software to define the number of regular channels to be converted in discontinuous mode, after receiving an external trigger.
000: 1 channel
001: 2 channels
.....
111: 8 channels

Bit 12 **JDISCEN**: Discontinuous mode on injected channels

This bit set and cleared by software to enable/disable discontinuous mode on injected group channels

0: Discontinuous mode on injected channels disabled

1: Discontinuous mode on injected channels enabled

Bit 11 **DISCEN**: Discontinuous mode on regular channels

This bit set and cleared by software to enable/disable Discontinuous mode on regular channels.

0: Discontinuous mode on regular channels disabled

1: Discontinuous mode on regular channels enabled

Bit 10 **JAUTO:** Automatic Injected Group conversion

This bit set and cleared by software to enable/disable automatic injected group conversion after regular group conversion.

0: Automatic injected group conversion disabled

1: Automatic injected group conversion enabled

Bit 9 **AWDSGL:** Enable the watchdog on a single channel in scan mode

This bit set and cleared by software to enable/disable the analog watchdog on the channel identified by the AWDCH[4:0] bits.

0: Analog watchdog enabled on all channels

1: Analog watchdog enabled on a single channel

Bit 8 **SCAN:** Scan mode

This bit is set and cleared by software to enable/disable Scan mode. In Scan mode, the inputs selected through the ADC_SQRx or ADC_JSQRx registers are converted.

0: Scan mode disabled

1: Scan mode enabled

Note: An EOC or JEOC interrupt is generated only on the end of conversion of the last channel if the corresponding EOCIE or JEOCIE bit is set

Bit 7 **JEOCIE**: Interrupt enable for injected channels

This bit is set and cleared by software to enable/disable the end of conversion interrupt for injected channels.

0: JEOC interrupt disabled

1: JEOC interrupt enabled. An interrupt is generated when the JEOC bit is set.

Bit 6 **AWDIE***:* Analog watchdog interrupt enable

This bit is set and cleared by software to enable/disable the analog watchdog interrupt. In Scan mode if the watchdog thresholds are crossed, scan is aborted only if this bit is enabled.

0: Analog watchdog interrupt disabled

1: Analog watchdog interrupt enabled

Bit 5 **EOCIE:** Interrupt enable for EOC

This bit is set and cleared by software to enable/disable the End of Conversion interrupt.

0: EOC interrupt disabled

1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

Bits 4:0 **AWDCH[4:0]**: *Analog watchdog channel select bits*

These bits are set and cleared by software. They select the input channel to be guarded by the Analog watchdog.

00000: ADC analog input Channel0
00001: ADC analog input Channel1
....
01111: ADC analog input Channel15
10000: ADC analog input Channel16
10001: ADC analog input Channel17
Other values reserved.

Note: ADC1 analog inputs Channel16 and Channel17 are internally connected to the temperature sensor and to V_{REFINT}, respectively.
ADC2 analog inputs Channel16 and Channel17 are internally connected to V_{SS}.
ADC3 analog inputs Channel9, Channel14, Channel15, Channel16 and Channel17 are connected to V_{SS}.

10.12.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
Res.								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw	Res.				rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bit 23 **TSVREFE**: Temperature sensor and V_{REFINT} enable

This bit is set and cleared by software to enable/disable the temperature sensor and V_{REFINT} channel. In devices with dual ADCs this bit is present only in ADC1.

0: Temperature sensor and V_{REFINT} channel disabled
1: Temperature sensor and V_{REFINT} channel enabled

Bit 22 **SWSTART**: Start Conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as conversion starts. It starts a conversion of a group of regular channels if SWSTART is selected as trigger event by the EXTSEL[2:0] bits.

0: Reset state
1: Starts conversion of regular channels

Bit 21 **JSWSTART**: Start Conversion of injected channels

This bit is set by software and cleared by software or by hardware as soon as the conversion starts. It starts a conversion of a group of injected channels (if JSWSTART is selected as trigger event by the JEXTSEL[2:0] bits.

0: Reset state
1: Starts conversion of injected channels

Bit 20 **EXTTRIG**: External Trigger Conversion mode for regular channels

This bit is set and cleared by software to enable/disable the external trigger used to start conversion of a regular channel group.
0: Conversion on external event disabled
1: Conversion on external event enabled

Bits 19:17 **EXTSEL[2:0]**: External event select for regular group

These bits select the external event used to trigger the start of conversion of a regular group:
For ADC1 and ADC2, the assigned triggers are:
000: Timer 1 CC1 event
001: Timer 1 CC2 event
010: Timer 1 CC3 event
011: Timer 2 CC2 event
100: Timer 3 TRGO event
101: Timer 4 CC4 event
110: EXTI line11/TIM8_TRGO event (TIM8_TRGO is available only in high-density devices)
111: SWSTART

For ADC3, the assigned triggers are:
000: Timer 3 CC1 event
001: Timer 2 CC3 event
010: Timer 1 CC3 event
011: Timer 8 CC1 event
100: Timer 8 TRGO event
101: Timer 5 CC1 event
110: Timer 5 CC3 event
111: SWSTART

Bit 16 Reserved, must be kept cleared.

Bit 15 **JEXTTRIG**: External trigger conversion mode for injected channels

This bit is set and cleared by software to enable/disable the external trigger used to start conversion of an injected channel group.
0: Conversion on external event disabled
1: Conversion on external event enabled

Bits 14:12 **JEXTSEL[2:0]**: External event select for injected group

These bits select the external event used to trigger the start of conversion of an injected group:

For ADC1 and ADC2 the assigned triggers are:

000: Timer 1 TRGO event
001: Timer 1 CC4 event
010: Timer 2 TRGO event
011: Timer 2 CC1 event
100: Timer 3 CC4 event
101: Timer 4 TRGO event
110: EXTI line15/TIM8_CC4 event (TIM8_CC4 is available only in High-density devices)
111: JSWSTART

For ADC3 the assigned triggers are:

000: Timer 1 TRGO event
001: Timer 1 CC4 event
010: Timer 4 CC3 event
011: Timer 8 CC2 event
100: Timer 8 CC4 event
101: Timer 5 TRGO event
110: Timer 5 CC4 event
111: JSWSTART

Bit 11 **ALIGN**: Data alignment

This bit is set and cleared by software. Refer to *Figure 25.*and *Figure 26.*

0: Right Alignment
1: Left Alignment

Bits 10:9 Reserved, must be kept cleared.

Bit 8 **DMA**: Direct memory access mode

This bit is set and cleared by software. Refer to the DMA controller chapter for more details.

0: DMA mode disabled
1: DMA mode enabled

Note: Only ADC1 and ADC3 can generate a DMA request.

Bits 7:4 Reserved, must be kept cleared.

Bit 3 **RSTCAL:** Reset calibration

This bit is set by software and cleared by hardware. It is cleared after the calibration registers are initialized.

0: Calibration register initialized.
1: Initialize calibration register.

Note: If RSTCAL is set when conversion is ongoing, additional cycles are required to clear the calibration registers.

Bit 2 **CAL:** A/D Calibration

This bit is set by software to start the calibration. It is reset by hardware after calibration is complete.

0: Calibration completed
1: Enable calibration

Bit 1 **CONT:** Continuous conversion

This bit is set and cleared by software. If set conversion takes place continuously till this bit is reset.

0: Single conversion mode

1: Continuous conversion mode

Bit 0 **ADON**: A/D converter ON / OFF

This bit is set and cleared by software. If this bit holds a value of zero and a 1 is written to it then it wakes up the ADC from Power Down state.

Conversion starts when this bit holds a value of 1 and a 1 is written to it. The application should allow a delay of t_{STAB} between power up and start of conversion. Refer to *Figure 21.*

0: Disable ADC conversion/calibration and go to power down mode.

1: Enable ADC and to start conversion

Note: If any other bit in this register apart from ADON is changed at the same time, then conversion is not triggered. This is to prevent triggering an erroneous conversion.

10.12.4 ADC sample time register 1 (ADC_SMPR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
Res.								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bits 23:0 **SMPx[2:0]:** Channel x Sample time selection

These bits are written by software to select the sample time individually for each channel. During sample cycles channel selection bits must remain unchanged.

000: 1.5 cycles

001: 7.5 cycles

010: 13.5 cycles

011: 28.5 cycles

100: 41.5 cycles

101: 55.5 cycles

110: 71.5 cycles

111: 239.5 cycles

Note: ADC1 analog inputs Channel16 and Channel17 are internally connected to the temperature sensor and to V_{REFINT}, respectively.

ADC2 analog input Channel16 and Channel17 are internally connected to V_{SS}.

ADC3 analog inputs Channel14, Channel15, Channel16 and Channel17 are connected to V_{SS}.

10.12.5 ADC sample time register 2 (ADC_SMPR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
Res.		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:0 **SMPx[2:0]:** Channel x Sample time selection

These bits are written by software to select the sample time individually for each channel. During sample cycles channel selection bits must remain unchanged.

000: 1.5 cycles
001: 7.5 cycles
010: 13.5 cycles
011: 28.5 cycles
100: 41.5 cycles
101: 55.5 cycles
110: 71.5 cycles
111: 239.5 cycles

Note: ADC3 analog input Channel9 is connected to V_{SS}.

10.12.6 ADC injected channel data offset register x (ADC_JOFRx)(x=1..4)

Address offset: 0x14-0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				JOFFSETx[11:0]											
Res.				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept cleared.

Bits 11:0 **JOFFSETx[11:0]**: Data offset for injected channel x

These bits are written by software to define the offset to be subtracted from the raw converted data when converting injected channels. The conversion result can be read from in the ADC_JDRx registers.

10.12.7 ADC watchdog high threshold register (ADC_HTR)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				HT[11:0]											
Res.				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept cleared.

Bits 11:0 **HT[11:0]:** Analog watchdog high threshold

These bits are written by software to define the high threshold for the analog watchdog.

10.12.8 ADC watchdog low threshold register (ADC_LTR)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				LT[11:0]											
Res				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept cleared.

Bits 11:0 **LT[11:0]** *Analog watchdog low threshold*

These bits are written by software to define the low threshold for the analog watchdog.

10.12.9 ADC regular sequence register 1 (ADC_SQR1)

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
Res.								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]					SQ14[4:0]					SQ13[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bits 23:20 **L[3:0]**: Regular channel sequence length

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

.....

1111: 16 conversions

Bits 19:15 **SQ16[4:0]**: 16th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 16th in the conversion sequence.

Bits 14:10 **SQ15[4:0]**: 15th conversion in regular sequence

Bits 9:5 **SQ14[4:0]**: 14th conversion in regular sequence

Bits 4:0 **SQ13[4:0]**: 13th conversion in regular sequence

10.12.10 ADC regular sequence register 2 (ADC_SQR2)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ12[4:0]					SQ11[4:0]					SQ10[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10_0	SQ9[4:0]					SQ8[4:0]					SQ7[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:26 **SQ12[4:0]**: *1*2th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 12th in the sequence to be converted.

Bits 24:20 **SQ11[4:0]**: 11th conversion in regular sequence

Bits 19:15 **SQ10[4:0]**: 10th conversion in regular sequence

Bits 14:10 **SQ9[4:0]:** 9th conversion in regular sequence

Bits 9:5 **SQ8[4:0]**: 8th conversion in regular sequence

Bits 4:0 **SQ7[4:0]**: 7th conversion in regular sequence

10.12.11 ADC regular sequence register 3 (ADC_SQR3)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0	SQ3[4:0]					SQ2[4:0]					SQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:25 **SQ6[4:0]**: 6th conversion in regular sequence
These bits are written by software with the channel number (0..17) assigned as the 6th in the sequence to be converted.

Bits 24:20 **SQ5[4:0]:** 5th conversion in regular sequence

Bits 19:15 **SQ4[4:0]**: 4th conversion in regular sequence

Bits 14:10 **SQ3[4:0]**: 3rd conversion in regular sequence

Bits 9:5 **SQ2[4:0]:** 2nd conversion in regular sequence

Bits 4:0 **SQ1[4:0]**: 1st conversion in regular sequence

10.12.12 ADC injected sequence register (ADC_JSQR)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										JL[1:0]		JSQ4[4:1]			
										rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSQ4_0	JSQ3[4:0]					JSQ2[4:0]					JSQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:22 Reserved, must be kept cleared.

Bits 21:20 **JL[1:0]**: *Injected Sequence length*

These bits are written by software to define the total number of conversions in the injected channel conversion sequence.

00: 1 conversion
01: 2 conversions
10: 3 conversions
11: 4 conversions

Bits 19:15 **JSQ4[4:0]**: *4th conversion in injected sequence*

These bits are written by software with the channel number (0..17) assigned as the 4th in the sequence to be converted.

Note: Unlike a regular conversion sequence, if JL[1:0] length is less than four, the channels are converted in a sequence starting from (4-JL). Example: ADC_JSQR[21:0] = 10 00011 00011 00111 00010 means that a scan conversion will convert the following channel sequence: 7, 3, 3. (not 2, 7, 3)

Bits 14:10 **JSQ3[4:0]:** *3rd conversion in injected sequence*

Bits 9:5 **JSQ2[4:0]**: *2nd conversion in injected sequence*

Bits 4:0 **JSQ1[4:0]**: *1st conversion in injected sequence*

10.12.13 ADC injected data register x (ADC_JDRx) (x= 1..4)

Address offset: 0x3C - 0x48

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JDATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept cleared.

Bits 15:0 **JDATA[15:0]**: Injected data

These bits are read only. They contain the conversion result from injected channel x. The data is left or right-aligned as shown in *Figure 25* and *Figure 26*.

10.12.14 ADC regular data register (ADC_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC2DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **ADC2DATA[15:0]:** ADC2 data

– In ADC1: In dual mode, these bits contain the regular data of ADC2. Refer to *Section 10.9: Dual ADC mode*

– In ADC2 and ADC3: these bits are not used

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read only. They contain the conversion result from the regular channels. The data is left or right-aligned as shown in *Figure 25* and *Figure 26*.

10.12.15 ADC register map

The following table summarizes the ADC registers.

Table 50. ADC register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**ADC_SR**	Reserved																											STRT	JSTRT	JEOC	EOC	AWD
	Reset value																												0	0	0	0	0
0x04	**ADC_CR1**	Reserved								AWDEN	JAWDEN	Reserved		DUALMOD [3:0]				DISC NUM [2:0]			JDISCEN	DISCEN	JAUTO	AWD SGL	SCAN	JEOC IE	AWDIE	EOCIE	AWDCH[4:0]				
	Reset value									0	0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**ADC_CR2**	Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL [2:0]			Reserved	JEXTTRIG	JEXTSEL [2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
	Reset value									0	0	0	0	0	0	0		0	0	0	0	0			0					0	0	0	0
0x0C	**ADC_SMPR1**	Sample time bits SMPx_x																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**ADC_SMPR2**	Sample time bits SMPx_x																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**ADC_JOFR1**	Reserved																				JOFFSET1[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	**ADC_JOFR2**	Reserved																				JOFFSET2[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**ADC_JOFR3**	Reserved																				JOFFSET3[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x20	**ADC_JOFR4**	Reserved																				JOFFSET4[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x24	**ADC_HTR**	Reserved																				HT[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x28	**ADC_LTR**	Reserved																				LT[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**ADC_SQR1**	Reserved								L[3:0]				Regular channel sequence SQx_x bits																			
	Reset value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 50. ADC register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x30	**ADC_SQR2**	Reserved		Regular channel sequence SQx_x bits																													
	Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x34	**ADC_SQR3**	Reserved		Regular channel sequence SQx_x bits																													
	Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x38	**ADC_JSQR**	Reserved										JL[1:0]		Injected channel sequence JSQx_x bits																			
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	**ADC_JDR1**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x40	**ADC_JDR2**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44	**ADC_JDR3**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x48	**ADC_JDR4**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x4C	**ADC_DR**	ADC2DATA[15:0]																Regular DATA[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

11 Digital-to-analog converter (DAC)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to high-density STM32F101xx and STM32F103xx devices only.

11.1 DAC introduction

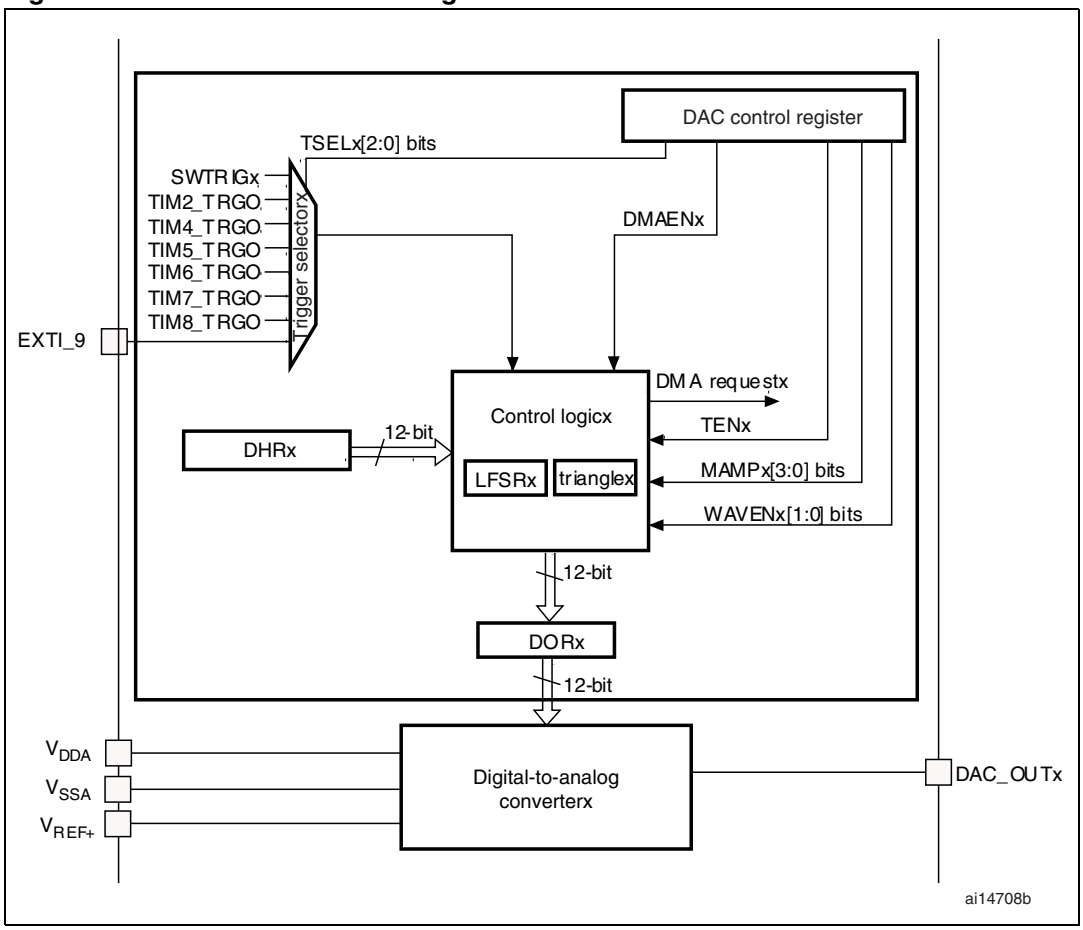
The DAC module is a 12-bit, voltage output digital-to-analog converter. The DAC can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. In 12-bit mode, the data could be left- or right-aligned. The DAC has two output channels, each with its own converter. In dual DAC channel mode, conversions could be done independently or simultaneously when both channels are grouped together for synchronous update operation. An input reference pin V_{REF+} is available for better resolution.

11.2 DAC main features

- Two DAC converters: one output channel each
- 8-bit or 12-bit monotonic output
- Left or right data alignment in 12-bit mode
- Synchronized update capability
- Noise-wave generation
- Triangular-wave generation
- Dual DAC channel independent or simultaneous conversions
- DMA capability for each channel
- External triggers for conversion
- Input voltage reference V_{REF+}

The block diagram of a DAC channel is shown in *Figure 38* and the pin description is given in *Table 51*.

Figure 38. DAC channel block diagram



Note:

Table 51. DAC pins

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the DAC, 2.4 V $\leq V_{REF+} \leq V_{DDA}$ (3.3 V)
V_{DDA}	Input, analog supply	Analog power supply
V_{SSA}	Input, analog supply ground	Ground for analog power supply
DAC_OUTx	Analog output signal	DAC channelx analog output

Note: Once the DAC channelx is enabled, the corresponding GPIO pin (PA4 or PA5) is automatically connected to the analog converter output (DAC_OUTx). In order to avoid parasitic consumption, the PA4 or PA5 pin should first be configured to analog (AIN).

11.3 DAC functional description

11.3.1 DAC channel enable

Each DAC channel can be powered on by setting its corresponding ENx bit in the DAC_CR register. The DAC channel is then enabled after a startup time t_{WAKEUP}.

Note: The ENx bit enables the analog DAC Channelx macrocell only. The DAC Channelx digital interface is enabled even if the ENx bit is reset.

11.3.2 DAC output buffer enable

The DAC integrates two output buffers that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. Each DAC channel output buffer can be enabled and disabled using the corresponding BOFFx bit in the DAC_CR register.

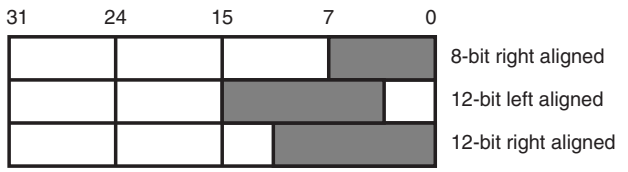
11.3.3 DAC data format

Depending on the selected configuration mode, the data has to be written in the specified register as described below:

- Single DAC channelx, there are three possibilities:
  - 8-bit right alignment: user has to load data into DAC_DHR8Rx [7:0] bits (stored into DHRx[11:4] bits)
  - 12-bit left alignment: user has to load data into DAC_DHR12Lx [15:4] bits (stored into DHRx[11:0] bits)
  - 12-bit right alignment: user has to load data into DAC_DHR12Rx [11:0] bits (stored into DHRx[11:0] bits)

Depending on the loaded DAC_DHRyyyx register, the data written by the user will be shifted and stored into the DHRx (Data Holding Registerx, that are internal non-memory-mapped registers). The DHRx register will then be loaded into the DORx register either automatically, by software trigger or by an external event trigger.

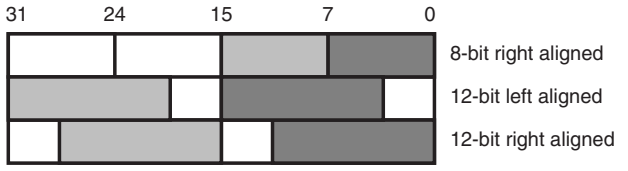
Figure 39. Data registers in single DAC channel mode



- Dual DAC channels, there are three possibilities:
  - 8-bit right alignment: data for DAC channel1 to be loaded into DAC_DHR8RD [7:0] bits (stored into DHR1[11:4] bits) and data for DAC channel2 to be loaded into DAC_DHR8RD [15:8] bits (stored into DHR2[11:4] bits)
  - 12-bit left alignment: data for DAC channel1 to be loaded into DAC_DHR12LD [15:4] bits (stored into DHR1[11:0] bits) and data for DAC channel2 to be loaded into DAC_DHR12LD [31:20] bits (stored into DHR2[11:0] bits)
  - 12-bit right alignment: data for DAC channel1 to be loaded into DAC_DHR12RD [11:0] bits (stored into DHR1[11:0] bits) and data for DAC channel2 to be loaded into DAC_DHR12LD [27:16] bits (stored into DHR2[11:0] bits)

Depending on the loaded DAC_DHRyyyD register, the data written by the user will be shifted and stored into the DHR1 and DHR2 (Data Holding Registers, that are internal non-memory-mapped registers). The DHR1 and DHR2 registers will then be loaded into the DOR1 and DOR2 registers, respectively, either automatically, by software trigger or by an external event trigger.

Figure 40. Data registers in dual DAC channel mode



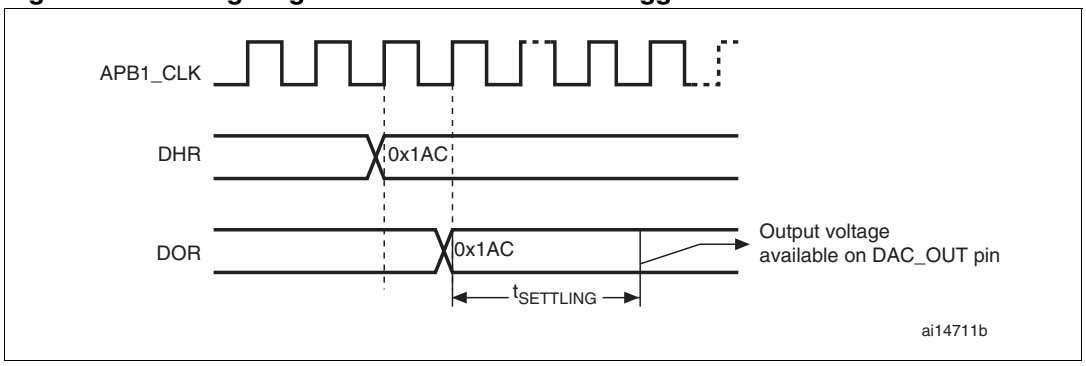
11.3.4 DAC conversion

The DAC_DORx cannot be written directly and any data transfer to the DAC channelx must be performed by loading the DAC_DHRx register (write on DAC_DHR8Rx, DAC_DHR12Lx, DAC_DHR12Rx, DAC_DHR8RD, DAC_DHR12LD or DAC_DHR12LD).

Data stored into the DAC_DHRx register are automatically transferred to the DAC_DORx register after one APB1 clock cycle, if no hardware trigger is selected (TENx bit in DAC_CR register is reset). However, when a hardware trigger is selected (TENx bit in DAC_CR register is set) and a trigger occurs, the transfer is performed three APB1 clock cycles later.

When DAC_DORx is loaded with the DAC_DHRx contents, the analog output voltage becomes available after a time of $t_{SETTLING}$ that depends on the power supply voltage and the analog output load.

Figure 41. Timing diagram for conversion with trigger disabled TEN = 0



11.3.5 DAC output voltage

Digital inputs are converted to output voltages on a linear conversion between 0 and V_{REF+}.

The analog output voltages on each DAC channel pin are determined by the following equation:

$$\text{DACoutput} = V_{REF} \times \frac{DOR}{4095}$$

11.3.6 DAC trigger selection

If the TENx control bit is set, conversion can then be triggered by an external event (timer counter, external interrupt line). The TSELx[2:0] control bits determine which out of 8 possible events will trigger conversion as shown in *Table 52*.

Table 52. External triggers

Source	Type	TSEL[2:0]
Timer 6 TRGO event	Internal signal from on-chip timers	000
Timer 8 TRGO event		001
Timer 7 TRGO event		010
Timer 5 TRGO event		011
Timer 2 TRGO event		100
Timer 4 TRGO event		101
EXTI line9	External pin	110
SWTRIG	Software control bit	111

Each time a DAC interface detects a rising edge on the selected timer TRGO output, or on the selected external interrupt line 9, the last data stored into the DAC_DHRx register is transferred into the DAC_DORx register. The DAC_DORx register is updated three APB1 cycles after the trigger occurs.

If the software trigger is selected, the conversion starts once the SWTRIG bit is set. SWTRIG is reset by hardware once the DAC_DORx register has been loaded with the DAC_DHRx register contents.

Note: 1 TSELx[2:0] bit cannot be changed when the ENx bit is set.

2 When software trigger is selected, it takes only one APB1 clock cycle for DAC_DHRx-to-DAC_DORx register transfer.

11.3.7 DMA request

Each DAC channel has a DMA capability. Two DMA channels are used to service DAC channel DMA requests.

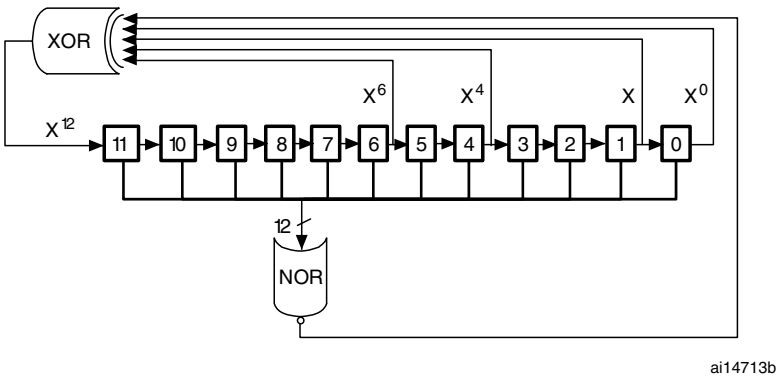
A DAC DMA request is generated when an external trigger (but not a software trigger) occurs while the DMAENx bit is set. The value of the DAC_DHRx register is then transferred to the DAC_DORx register.
In dual mode, if both DMAENx bits are set, two DMA requests are generated. If only one DMA request is needed, you should set only the corresponding DMAENx bit. In this way, the application can manage both DAC channels in dual mode by using one DMA request and a unique DMA channel.

The DAC DMA request is not queued so that if a second external trigger arrives before the acknowledgement of the last request, then the new request will not be serviced and no error is reported

11.3.8 Noise generation

In order to generate a variable-amplitude pseudonoise, a Linear Feedback Shift Register is available. The DAC noise generation is selected by setting WAVEx[1:0] to “01”. The preloaded value in the LFSR is 0xAAA. This register is updated, three APB1 clock cycles after each trigger event, following a specific calculation algorithm.

Figure 42. DAC LFSR register calculation algorithm

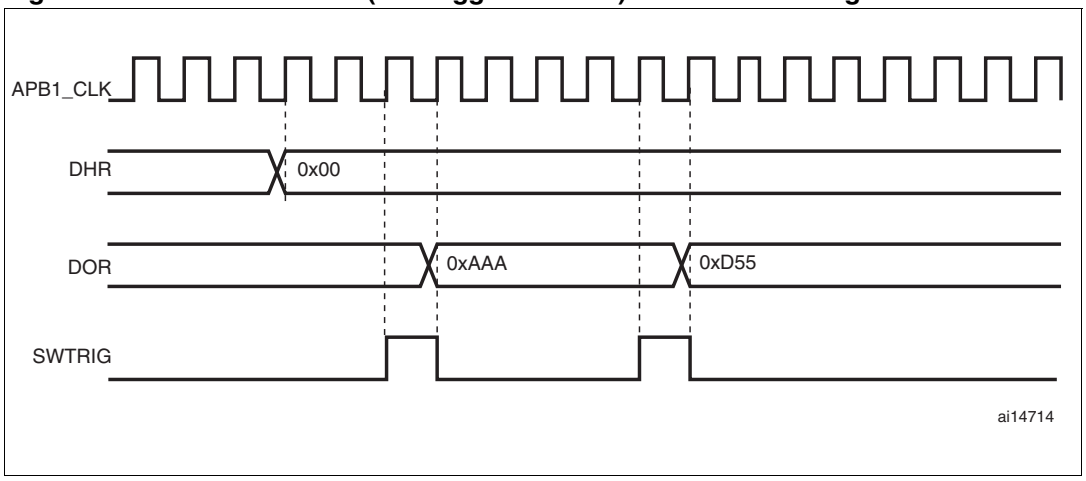


The LFSR value, that may be masked partially or totally by means of the MAMPx[3:0] bits in the DAC_CR register, is added up to the DAC_DHRx contents without overflow and this value is then stored into the DAC_DORx register.

If LFSR is 0x0000, a ‘1’ is injected into it (antilock-up mechanism).

It is possible to reset LFSR wave generation by resetting the WAVEx[1:0] bits.

Figure 43. DAC conversion (SW trigger enabled) with LFSR wave generation



Note: DAC trigger must be enabled for noise generation, by setting the TENx bit in the DAC_CR register.

11.3.9 Triangle-wave generation

It is possible to add a small-amplitude triangular waveform on a DC or slowly varying signal. DAC triangle-wave generation is selected by setting WAVEx[1:0] to “10”. The amplitude is configured through the MAMPx[3:0] bits in the DAC_CR register. An internal triangle counter is incremented three APB1 clock cycles after each trigger event. The value of this counter is then added to the DAC_DHRx register without overflow and the sum is stored into the DAC_DORx register. The triangle counter is incremented while the value transferred into the DAC_DORx register is less then the maximum amplitude defined by the MAMPx[3:0] bits. Once the configured amplitude is reached, the counter is decremented down to 0, then incremented again and so on.

It is possible to reset triangle wave generation by resetting WAVEx[1:0] bits.

Figure 44. DAC triangle wave generation

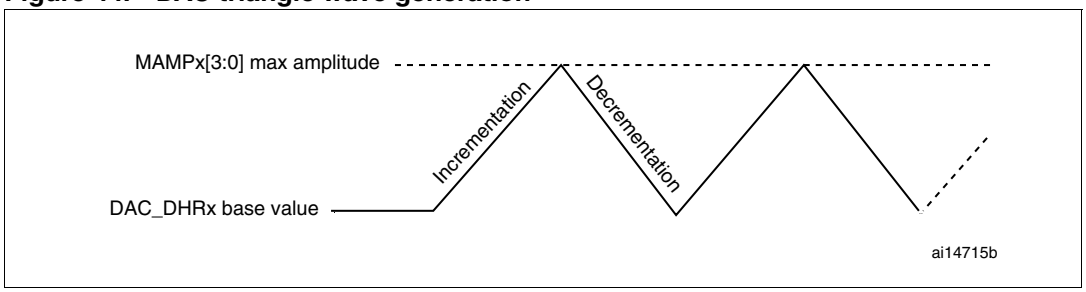
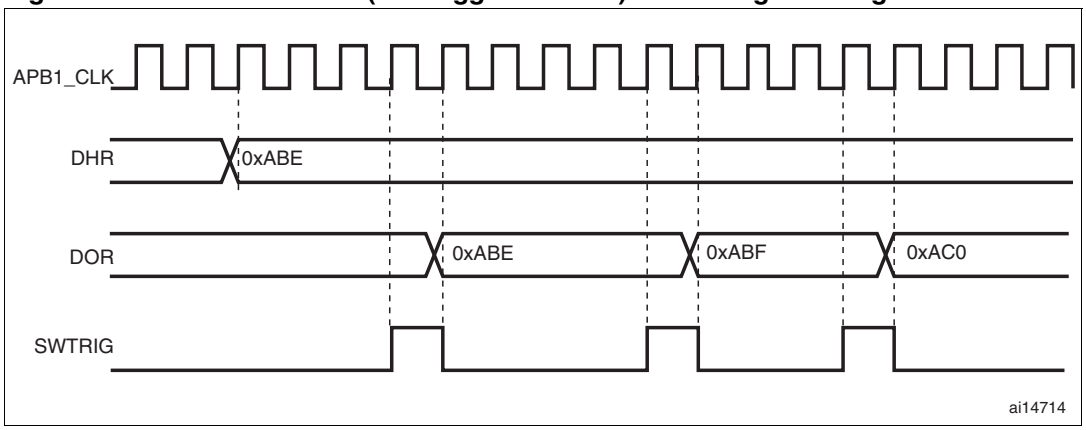


Figure 45. DAC conversion (SW trigger enabled) with triangle wave generation



Note: 1 DAC trigger must be enabled for noise generation, by setting the TENx bit in the DAC_CR register.

2 MAMPx[3:0] bits must be configured before enabling the DAC, otherwise they cannot be changed.

11.4 Dual DAC channel conversion

To efficiently use the bus bandwidth in applications that require the two DAC channels at the same time, three dual registers are implemented: DHR8RD, DHR12RD and DHR12LD. A unique register access is then required to drive both DAC channels at the same time.

Eleven possible conversion modes are possible using the two DAC channels and these dual registers. All the conversion modes can nevertheless be obtained using separate DHRx registers if needed.

All modes are described in the paragraphs below.

11.4.1 Independent trigger without wave generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the DHR1 register is transferred into DAC_DOR1 (three APB1 clock cycles later).

When a DAC channel2 trigger arrives, the DHR2 register is transferred into DAC_DOR2 (three APB1 clock cycles later).

11.4.2 Independent trigger with same LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DHR12RD, DHR12LD or DHR8RD)

When a DAC channel1 trigger arrives, the LFSR1 counter, with the same mask, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). Then the LFSR1 counter is updated.

When a DAC channel2 trigger arrives, the LFSR2 counter, with the same mask, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). Then the LFSR2 counter is updated.

11.4.3 Independent trigger with different LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and set different LFSR masks values in the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the LFSR1 counter, with the mask configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). Then the LFSR1 counter is updated.

When a DAC channel2 trigger arrives, the LFSR2 counter, with the mask configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). Then the LFSR2 counter is updated.

11.4.4 Independent trigger with same triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and the same maximum amplitude value in the MAMPx[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the DAC channel1 triangle counter, with the same triangle amplitude, is added to the DHR1 register and the sum is transferred into

DAC_DOR1 (three APB1 clock cycles later). The DAC channel1 triangle counter is then updated.

When a DAC channel2 trigger arrives, the DAC channel2 triangle counter, with the same triangle amplitude, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The DAC channel2 triangle counter is then updated.

11.4.5 Independent trigger with different triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and set different maximum amplitude values in the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the DAC channel1 triangle counter, with a triangle amplitude configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The DAC channel1 triangle counter is then updated.

When a DAC channel2 trigger arrives, the DAC channel2 triangle counter, with a triangle amplitude configured by MAMP2[3:0], is added to the DHR2 register part and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The DAC channel2 triangle counter is then updated.

11.4.6 Simultaneous software start

To configure the DAC in this conversion mode, the following sequence is required:

- Load the dual DAC channel data to the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

In this configuration, one APB1 clock cycle later, the DHR1 and DHR2 registers are transferred into DAC_DOR1 and DAC_DOR2, respectively.

11.4.7 Simultaneous trigger without wave generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Load the dual DAC channel data to the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the DHR1 and DHR2 registers are transferred into DAC_DOR1 and DAC_DOR2, respectively (after three APB1 clock cycles).

11.4.8 Simultaneous trigger with same LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
- Load the dual DAC channel data to the desired DHR register (DHR12RD, DHR12LD or DHR8RD)

When a trigger arrives, the LFSR1 counter, with the same mask, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The LFSR1 counter is then updated. At the same time, the LFSR2 counter, with the same mask, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The LFSR2 counter is then updated.

11.4.9 Simultaneous trigger with different LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and set different LFSR masks values using the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the LFSR1 counter, with the mask configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The LFSR1 counter is then updated.
At the same time, the LFSR2 counter, with the mask configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The LFSR2 counter is then updated.

11.4.10 Simultaneous trigger with same triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and the same maximum amplitude value using the MAMPx[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the DAC channel1 triangle counter, with the same triangle amplitude, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The DAC channel1 triangle counter is then updated.
At the same time, the DAC channel2 triangle counter, with the same triangle amplitude, is

added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The DAC channel2 triangle counter is then updated.

11.4.11 Simultaneous trigger with different triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and set different maximum amplitude values in the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the DAC channel1 triangle counter, with a triangle amplitude configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). Then the DAC channel1 triangle counter is updated.
At the same time, the DAC channel2 triangle counter, with a triangle amplitude configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). Then the DAC channel2 triangle counter is updated.

11.5 DAC registers

11.5.1 DAC control register (DAC_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			DMA EN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			DMA EN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:29 Reserved.

Bit 28 **DMAEN2**: DAC channel2 DMA enable

This bit is set and cleared by software.
0: DAC channel2 DMA mode disabled
1: DAC channel2 DMA mode enabled

Bit 27:24 **MAMP2[3:0]**: DAC channel2 Mask/Amplitude selector

These bits are written by software to select mask in wave generation mode or amplitude in triangle generation mode.
0000: Unmask bit0 of LFSR/ Triangle Amplitude equal to 1
0001: Unmask bits[1:0] of LFSR/ Triangle Amplitude equal to 3
0010: Unmask bits[2:0] of LFSR/ Triangle Amplitude equal to 7
0011: Unmask bits[3:0] of LFSR/ Triangle Amplitude equal to 15
0100: Unmask bits[4:0] of LFSR/ Triangle Amplitude equal to 31
0101: Unmask bits[5:0] of LFSR/ Triangle Amplitude equal to 63
0110: Unmask bits[6:0] of LFSR/ Triangle Amplitude equal to 127
0111: Unmask bits[7:0] of LFSR/ Triangle Amplitude equal to 255
1000: Unmask bits[8:0] of LFSR/ Triangle Amplitude equal to 511
1001: Unmask bits[9:0] of LFSR/ Triangle Amplitude equal to 1023
1010: Unmask bits[10:0] of LFSR/ Triangle Amplitude equal to 2047
≥ 1011: Unmask bits[11:0] of LFSR/ Triangle Amplitude equal to 4095

Bit 23:22 **WAVE2[1:0]**: DAC channel2 noise/triangle wave generation enable

These bits are set/reset by software.
00: wave generation disabled
01: Noise wave generation enabled
1x: Triangle wave generation enabled

Note: only used if bit TEN2 = 1 (DAC channel2 trigger enabled)

Bits 21:19 **TSEL2[2:0]**: DAC channel2 Trigger selection

These bits select the external event used to trigger DAC channel2
000: Timer 6 TRGO event
001: Timer 8 TRGO event
010: Timer 7 TRGO event
011: Timer 5 TRGO event
100: Timer 2 TRGO event
101: Timer 4 TRGO event
110: External line9
111: Software trigger

Note: only used if bit TEN2 = 1 (DAC channel2 trigger enabled)

Bit 18 **TEN2**: DAC channel2 Trigger enable

This bit set and cleared by software to enable/disable DAC channel2 trigger
0: DAC channel2 trigger disabled and data written into DAC_DHRx register is transferred one APB1 clock cycle later to the DAC_DOR2 register.
1: DAC channel2 trigger enabled and data transfer from DAC_DHRx register is transferred three APB1 clock cycles later to the DAC_DOR2 register.

Note: When software trigger is selected, it takes only one APB1 clock cycle for DAC_DHRx to DAC_DOR2 register transfer.

Bit 17 **BOFF2**: DAC channel2 output buffer disable

This bit set and cleared by software to enable/disable DAC channel2 output buffer.
0: DAC channel2 output buffer enabled
1: DAC channel2 output buffer disabled

Bit 16 **EN2**: DAC channel2 enable

This bit set and cleared by software to enable/disable DAC channel2.
0: DAC channel2 disabled
1: DAC channel2 enabled

Bits 15:13 Reserved.

Bit 12 **DMAEN1**: DAC channel1 DMA enable

This bit is set and cleared by software.
0: DAC channel1 DMA mode disabled
1: DAC channel1 DMA mode enabled

Bits 11:8 **MAMP1[3:0]**: DAC channel1 Mask/Amplitude selector

These bits are written by software to select mask in wave generation mode or amplitude in triangle generation mode.
0000: Unmask bit0 of LFSR/ Triangle Amplitude equal to 1
0001: Unmask bits[1:0] of LFSR/ Triangle Amplitude equal to 3
0010: Unmask bits[2:0] of LFSR/ Triangle Amplitude equal to 7
0011: Unmask bits[3:0] of LFSR/ Triangle Amplitude equal to 15
0100: Unmask bits[4:0] of LFSR/ Triangle Amplitude equal to 31
0101: Unmask bits[5:0] of LFSR/ Triangle Amplitude equal to 63
0110: Unmask bits[6:0] of LFSR/ Triangle Amplitude equal to 127
0111: Unmask bits[7:0] of LFSR/ Triangle Amplitude equal to 255
1000: Unmask bits[8:0] of LFSR/ Triangle Amplitude equal to 511
1001: Unmask bits[9:0] of LFSR/ Triangle Amplitude equal to 1023
1010: Unmask bits[10:0] of LFSR/ Triangle Amplitude equal to 2047
≥ 1011: Unmask bits[11:0] of LFSR/ Triangle Amplitude equal to 4095

Bits 7:6 **WAVE1[1:0]**: DAC channel1 noise/triangle wave generation enable

These bits are set/reset by software.
00: wave generation disabled
01: Noise wave generation enabled
1x: Triangle wave generation enabled
Note: only used if bit TEN1 = 1 (DAC channel1 trigger enabled)

Bits 5:3 **TSEL1[2:0]**: DAC channel1 Trigger selection

These bits select the external event used to trigger DAC channel1
000: Timer 6 TRGO event
001: Timer 83 TRGO event
010: Timer 7 TRGO event
011: Timer 5 TRGO event
100: Timer 2 TRGO event
101: Timer 4 TRGO event
110: External line9
111: Software trigger
Note: only used if bit TEN1 = 1 (DAC channel1 trigger enabled)

Bit 2 **TEN1**: DAC channel1 Trigger enable

This bit set and cleared by software to enable/disable DAC channel1 trigger

0: DAC channel1 trigger disabled and data written into DAC_DHRx register is transferred one APB1 clock cycle later to the DAC_DOR1 register.

1: DAC channel1 trigger enabled and data transfer from DAC_DHRx register is transferred three APB1 clock cycles later to the DAC_DOR1 register.

Note: When software trigger is selected, it takes only one APB1 clock cycle for DAC_DHRx to DAC_DOR1 register transfer.

Bit 1 **BOFF1**: DAC channel1 output buffer disable

This bit set and cleared by software to enable/disable DAC channel1 output buffer.

0: DAC channel1 output buffer enabled

1: DAC channel1 output buffer disabled

Bit 0 **EN1**: DAC channel1 enable

This bit set and cleared by software to enable/disable DAC channel1.

0: DAC channel1 disabled

1: DAC channel1 enabled

11.5.2 DAC software trigger register (DAC_SWTRIGR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														SWTRIG2	SWTRIG1
														w	w

Bits 31:2 Reserved.

Bit 1 **SWTRIG2**: DAC channel2 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

0: Software trigger disabled

1: Software trigger enabled

Note: This bit is reset by hardware (one APB1 clock cycle later) once the DAC_DHR2 register value is loaded to the DAC_DOR2 register.

Bit 0 **SWTRIG1**: DAC channel1 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

0: Software trigger disabled

1: Software trigger enabled

Note: This bit is reset by hardware (one APB1 clock cycle later) once the DAC_DHR1 register value is loaded to the DAC_DOR1 register.

11.5.3 DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DACC1DHR[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved.

Bit 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit Right aligned data
These bits are written by software which specify 12-bit data for DAC channel1.

11.5.4 DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC1DHR[11:0]												Reserved			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				

Bits 31:16 Reserved.

Bit 15:4 **DACC1DHR[11:0]**: *DAC channel1 12-bit Left aligned data*
These bits are written by software which specify 12-bit data for DAC channel1.

Bits 3:0 Reserved.

11.5.5 DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DACC1DHR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved.

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit Right aligned data
These bits are written by software which specify 8-bit data for DAC channel1.

11.5.6 DAC channel2 12-bit right aligned data holding register (DAC_DHR12R2)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DACC2DHR[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved.

Bits 11:0 **DACC2DHR[11:0]**: DAC channel2 12-bit Right aligned data
These bits are written by software which specify 12-bit data for DAC channel2.

11.5.7 DAC channel2 12-bit left aligned data holding register (DAC_DHR12L2)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[11:0]												Reserved			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				

Bits 31:16 Reserved.

Bits 15:4 **DACC2DHR[11:0]**: DAC channel2 12-bit Left aligned data
These bits are written by software which specify 12-bit data for DAC channel2.

Bits 3:0 Reserved.

11.5.8 DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DACC2DHR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved.

Bits 7:0 **DACC2DHR[7:0]**: *DAC channel2 8-bit Right aligned data*
These bits are written by software which specify 8-bit data for DAC channel2.

11.5.9 Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				DACC2DHR[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DACC1DHR[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28 Reserved.

Bits 27:16 **DACC2DHR[11:0]**: DAC channel2 12-bit Right aligned data
These bits are written by software which specify 12-bit data for DAC channel2.

Bits 15:12 Reserved.

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit Right aligned data
These bits are written by software which specify 12-bit data for DAC channel1.

11.5.10 DUAL DAC 12-bit left aligned data holding register (DAC_DHR12LD)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DACC2DHR[11:0]												Reserved			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC1DHR[11:0]												Reserved			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				

Bits 31:20 **DACC2DHR[11:0]**: DAC channel2 12-bit Left aligned data
These bits are written by software which specify 12-bit data for DAC channel2.

Bits 19:16 Reserved.

Bits 15:4 **DACC1DHR[11:0]**: DAC channel1 12-bit Left aligned data
These bits are written by software which specify 12-bit data for DAC channel1.

Bits 3:0 Reserved.

11.5.11 DUAL DAC 8-bit right aligned data holding register (DAC_DHR8RD)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[7:0]								DACC1DHR[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved.

Bits 15:8 **DACC2DHR[7:0]**: DAC channel2 8-bit Right aligned data
These bits are written by software which specify 8-bit data for DAC channel2.

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit Right aligned data
These bits are written by software which specify 8-bit data for DAC channel1.

11.5.12 DAC channel1 data output register (DAC_DOR1)

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DACC1DOR[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:12 Reserved.

Bit 11:0 **DACC1DOR[11:0]**: DAC channel1 data output
These bits are read only, they contain data output for DAC channel1.

11.5.13 DAC channel2 data output register (DAC_DOR2)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DACC2DOR[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:12 Reserved.

Bit 11:0 **DACC2DOR[11:0]**: DAC channel2 data output
These bits are read only, they contain data output for DAC channel2.

11.5.14 DAC register map

The following table summarizes the DAC registers.

Table 53. DAC register map

Address offset	Name	31																															0
0x00	DAC_CR	Reserved			DMAEN2	MAMP2[3:0]				WAVE2[2:0]		TSEL2[2:0]			TEN2	BOFF2	EN2	Reserved			DMAEN1	MAMP1[3:0]				WAVE1[2:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
0x04	DAC_SWTRIGR	Reserved																														SWTRIG2	SWTRIG1
0x08	DAC_DHR12R1	Reserved																			DACC1DHR[11:0]												
0x0C	DAC_DHR12L1	Reserved																DACC1DHR[11:0]														Reserved	
0x10	DAC_DHR8R1	Reserved																								DACC1DHR[7:0]							
0x14	DAC_DHR12R2	Reserved																			DACC2DHR[11:0]												
0x18	DAC_DHR12L2	Reserved																DACC2DHR[11:0]														Reserved	
0x1C	DAC_DHR8R2	Reserved																								DACC2DHR[7:0]							
0x20	DAC_DHR12RD	Reserved				DACC2DHR[11:0]												Reserved			DACC1DHR[11:0]												
0x24	DAC_DHR12LD	DACC2DHR[11:0]													Reserved			DACC1DHR[11:0]														Reserved	
0x28	DAC_DHR8RD	Reserved																DACC2DHR[7:0]								DACC1DHR[7:0]							
0x2C	DAC_DOR1	Reserved																			DACC1DOR[11:0]												
0x30	DAC_DOR2	Reserved																			DACC2DOR[11:0]												

Note: Refer to Table 1 on page 35 for the register boundary addresses.

12 Advanced-control timers (TIM1&TIM8)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

Low- and medium-density STM32F103xx devices contain one advanced-control timer (TIM1) whereas high-density STM32F103xx devices feature two advance-control timers (TIM1 and TIM8).

12.1 TIM1&TIM8 introduction

The advanced-control timers (TIM1&TIM8) consist of a 16-bit auto-reload counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The advanced-control (TIM1&TIM8) and general-purpose (TIMx) timers are completely independent, and do not share any resources. They can be synchronized together as described in *Section 12.3.20*.

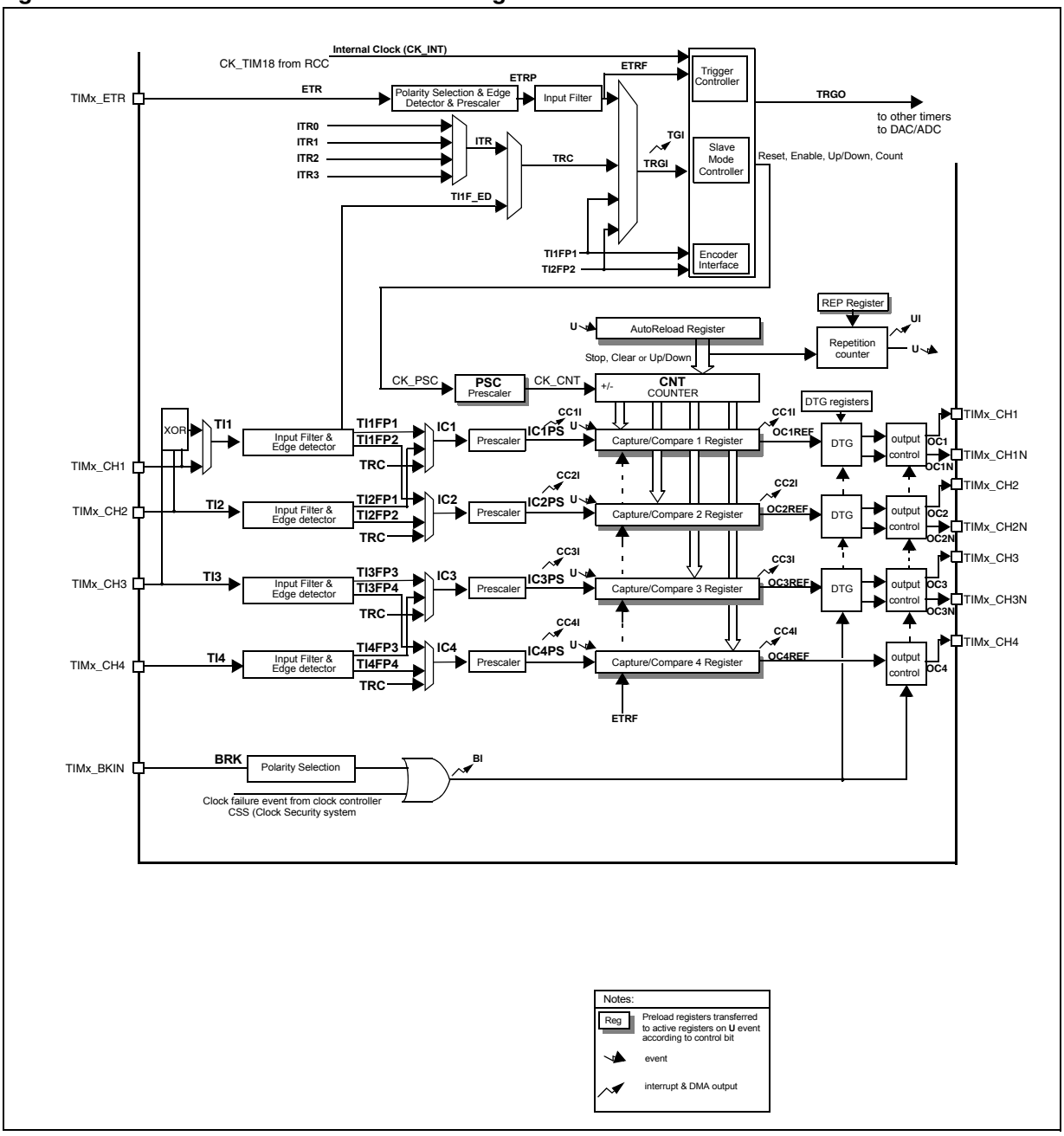
12.2 TIM1&TIM8 main features

TIM1&TIM8 timer features include:

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65535.
- Up to 4 independent channels for:
  - Input Capture
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time
- Synchronization circuit to control the timer with external signals and to interconnect several timers together.
- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- Break input to put the timer’s output signals in reset state or in a known state.

- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
  - Break input
- Supports incremental (quadrature) encoder and hall-sensor cicuitry for positioning purposes
- Trigger intput for external clock or cycle-by-cycle current management

Figure 46. Advanced-control timer block diagram



12.3 TIM1&TIM8 functional description

12.3.1 Time-base unit

The main block of the programmable advanced-control timer is a 16-bit counter with its related auto-reload register. The counter can count up, down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC)
- Auto-Reload Register (TIMx_ARR)
- Repetition Counter Register (TIMx_RCR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detailed for each configuration.

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx_CR1 register.

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It 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.

Figure 48 and *Figure 49* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

Figure 47. Counter timing diagram with prescaler division change from 1 to 2

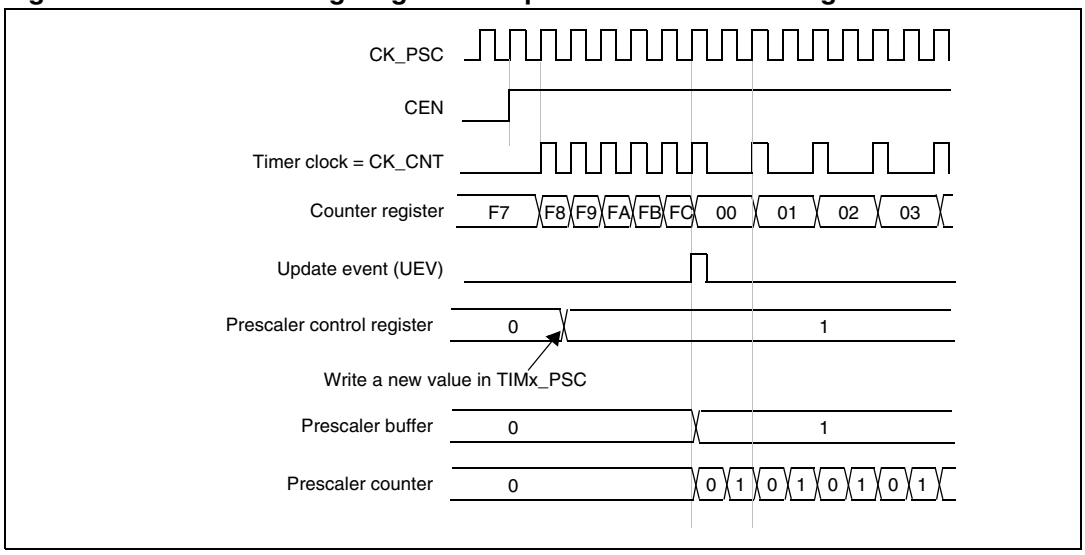
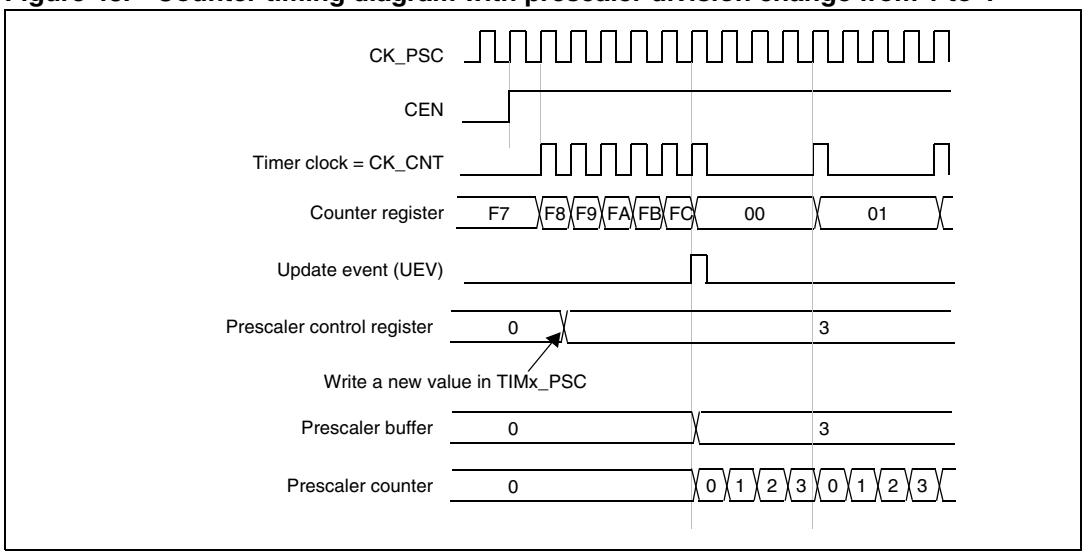


Figure 48. Counter timing diagram with prescaler division change from 1 to 4



12.3.2 Counter modes

Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

If the repetition counter is used, the update event (UEV) is generated after upcounting is repeated for the number of times programmed in the repetition counter register (TIMx_RCR). Else the update event is generated at each counter overflow.

Setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event.

The UEV event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the

preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The repetition counter is reloaded with the content of TIMx_RCR register,
- The auto-reload shadow register is updated with the preload value (TIMx_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 49. Counter timing diagram, internal clock divided by 1

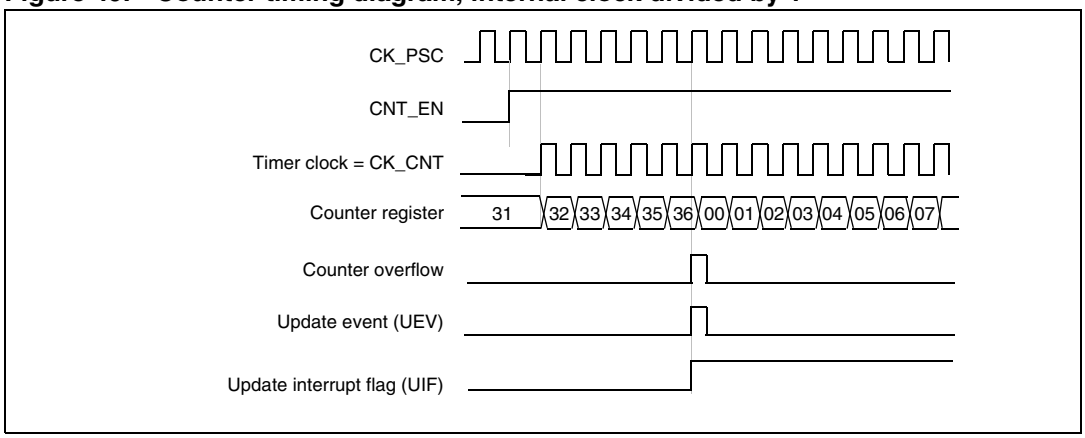


Figure 50. Counter timing diagram, internal clock divided by 2

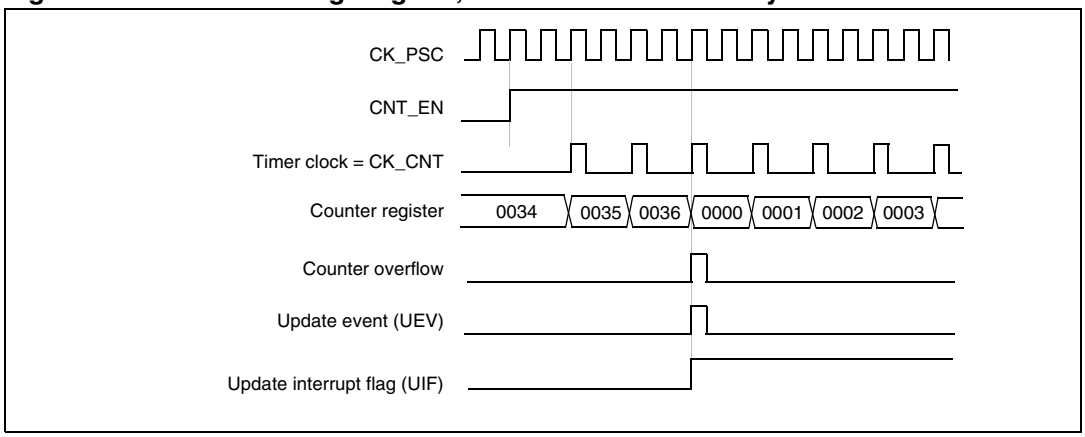


Figure 51. Counter timing diagram, internal clock divided by 4

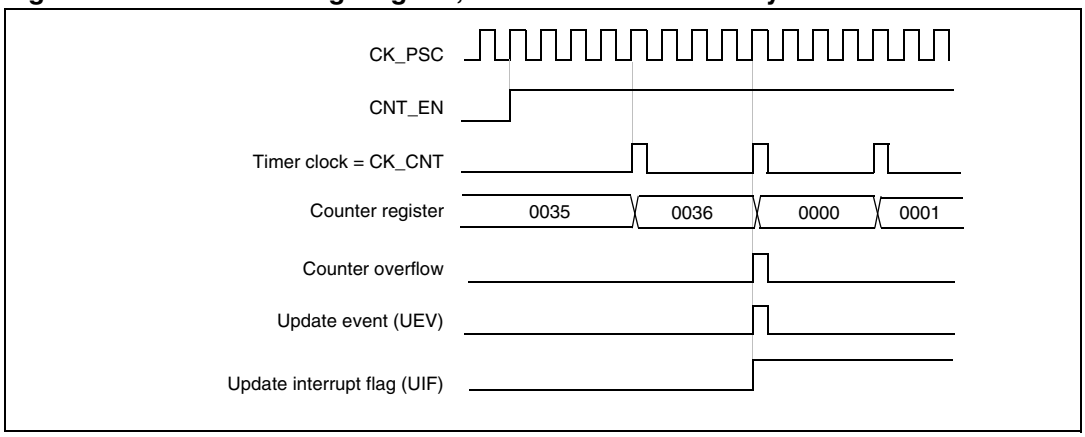


Figure 52. Counter timing diagram, internal clock divided by N

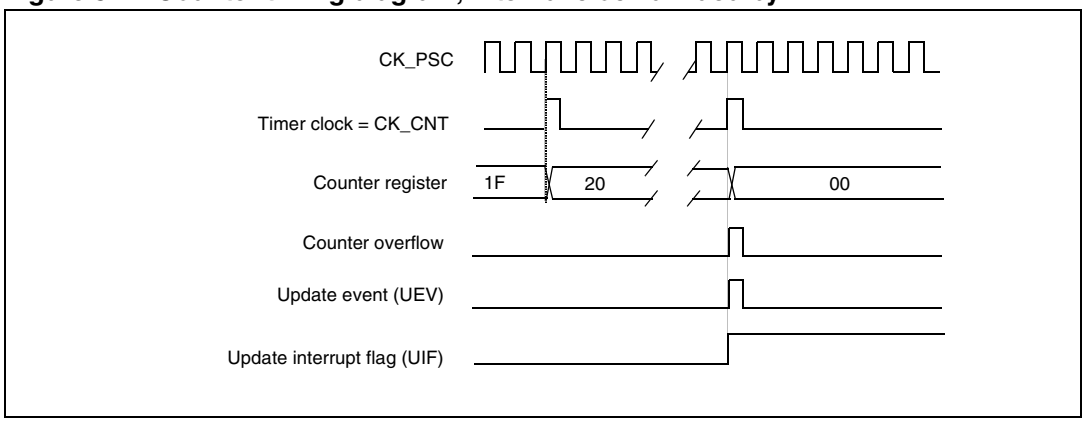


Figure 53. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded)

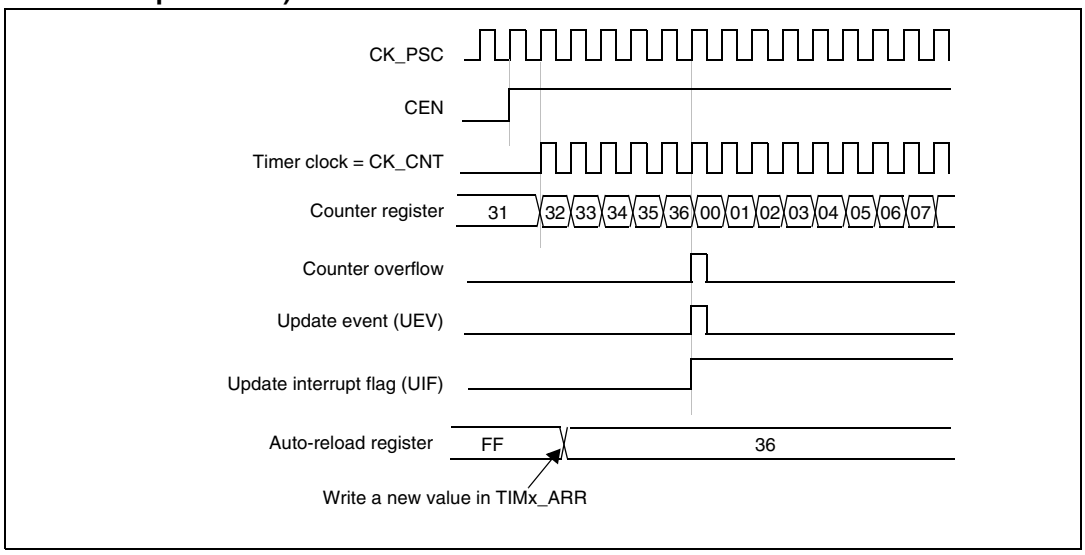
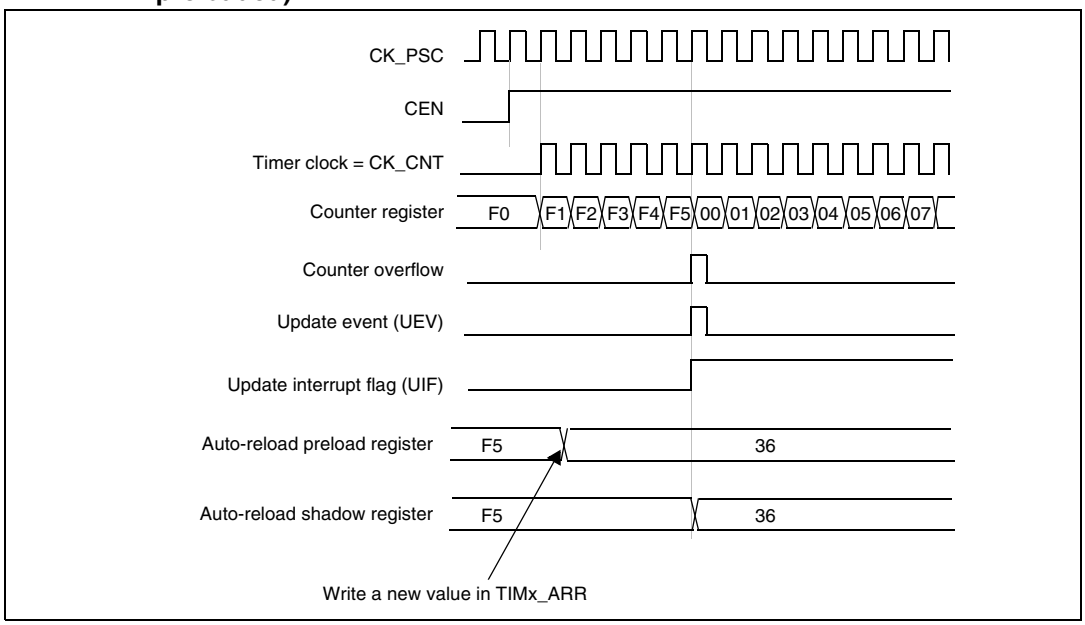


Figure 54. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)



Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

If the repetition counter is used, the update event (UEV) is generated after downcounting is repeated for the number of times programmed in the repetition counter register (TIMx_RCR). Else the update event is generated at each counter underflow.

Setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The repetition counter is reloaded with the content of TIMx_RCR register
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register)
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 55. Counter timing diagram, internal clock divided by 1

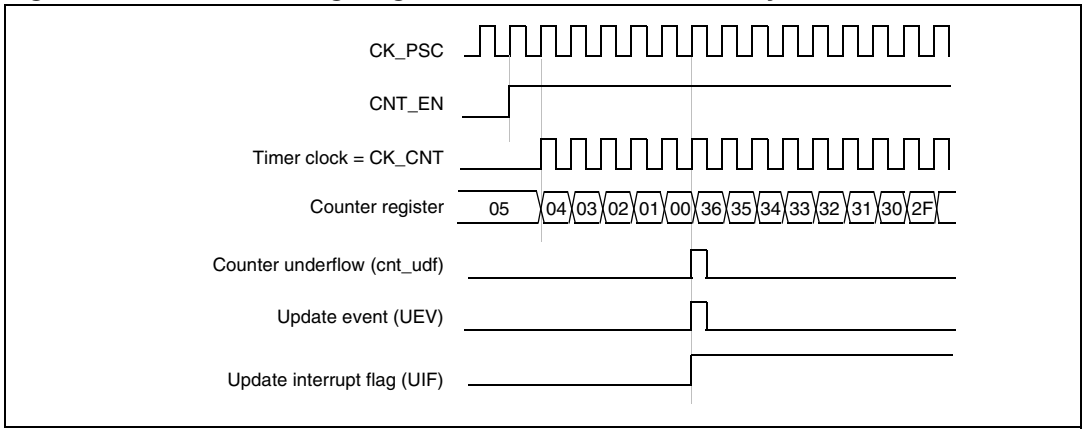


Figure 56. Counter timing diagram, internal clock divided by 2

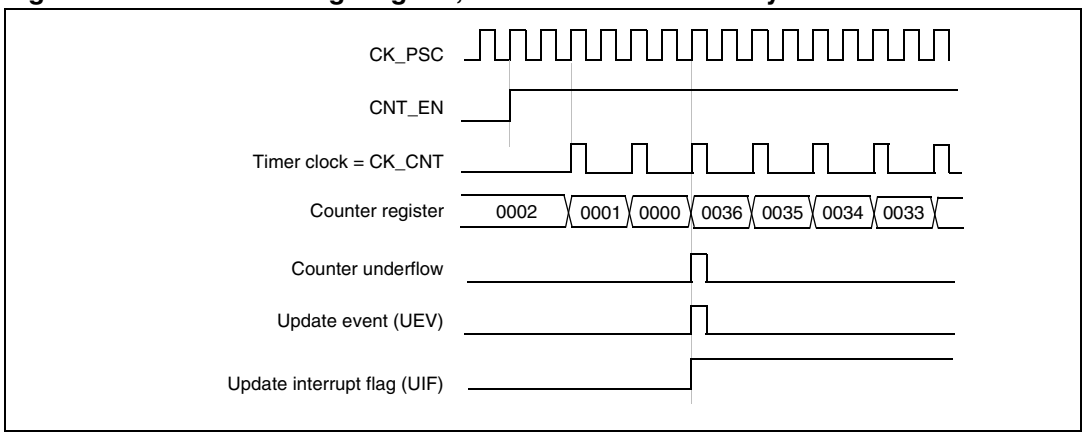


Figure 57. Counter timing diagram, internal clock divided by 4

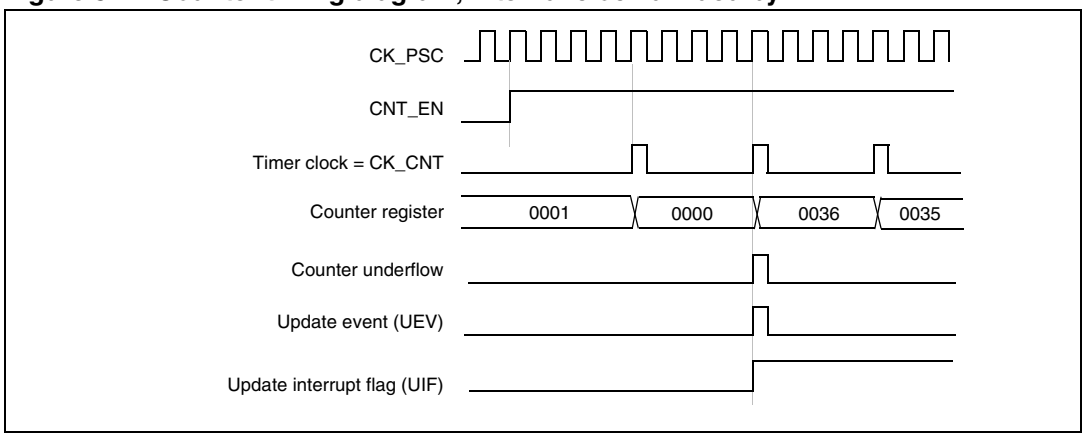


Figure 58. Counter timing diagram, internal clock divided by N

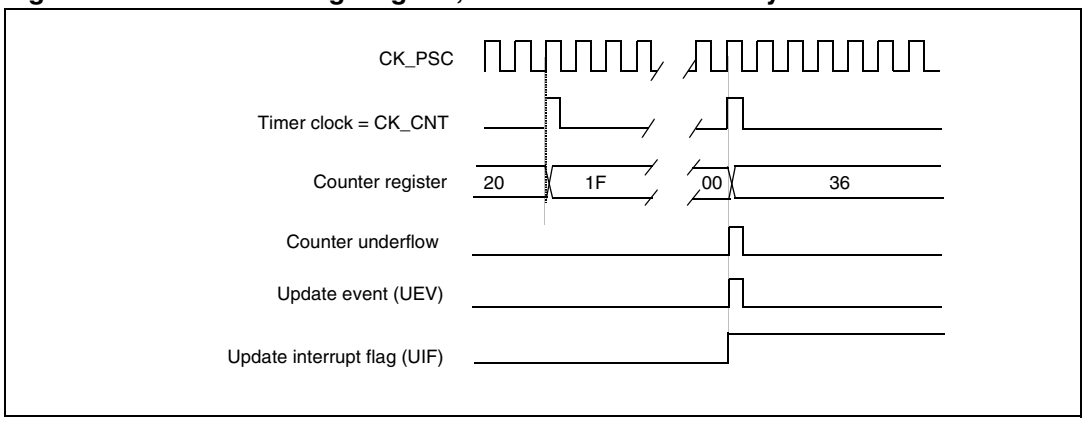
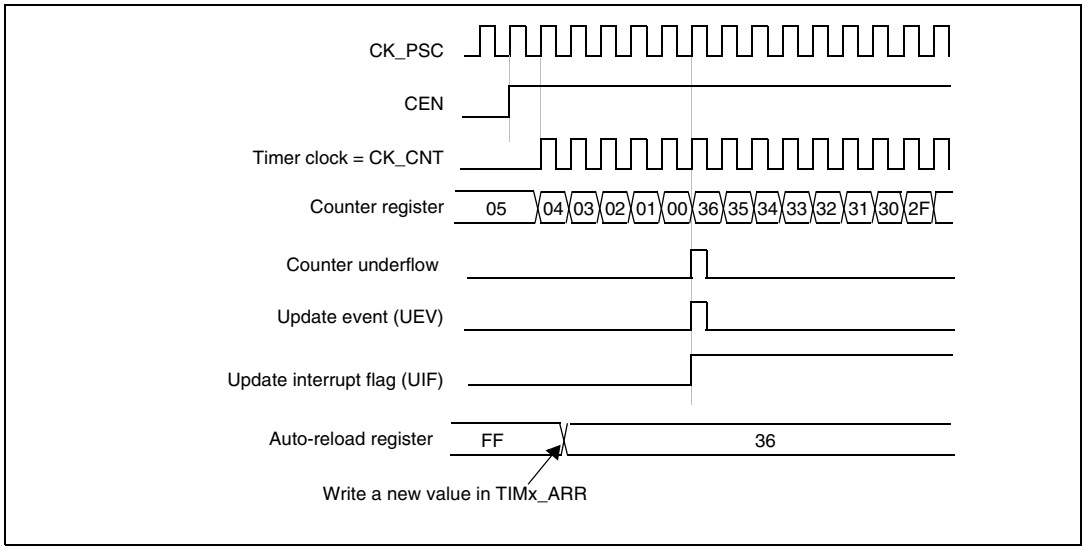


Figure 59. Counter timing diagram, update event when repetition counter is not used



Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

In this mode, the DIR direction bit in the TIMx_CR1 register cannot be written. It is updated by hardware and gives the current direction of the counter.

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The UEV update event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

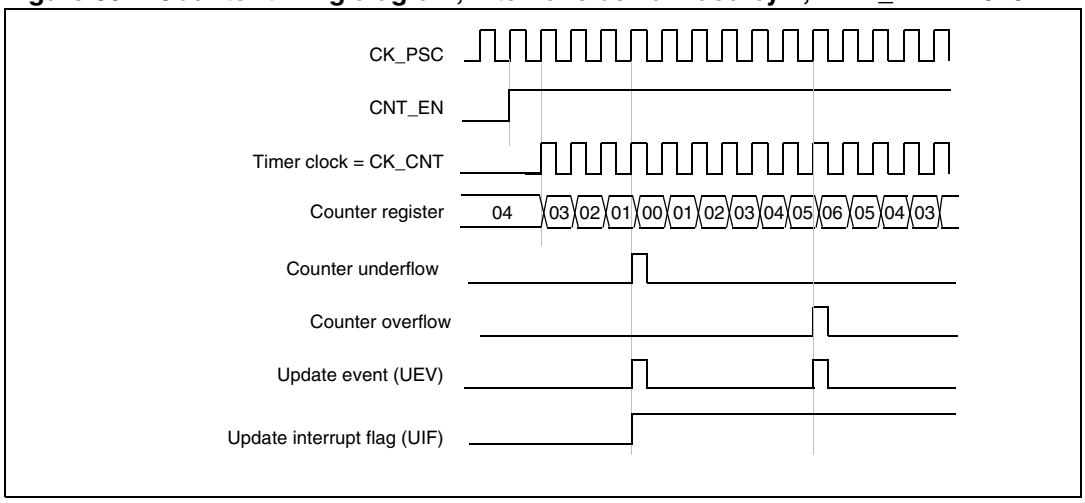
In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an UEV update event but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The repetition counter is reloaded with the content of TIMx_RCR register
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register)
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

Figure 60. Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6



1. Here, center-aligned mode 1 is used (for more details refer to *Section 12.4: TIM1&TIM8 registers on page 240*).

Figure 61. Counter timing diagram, internal clock divided by 2

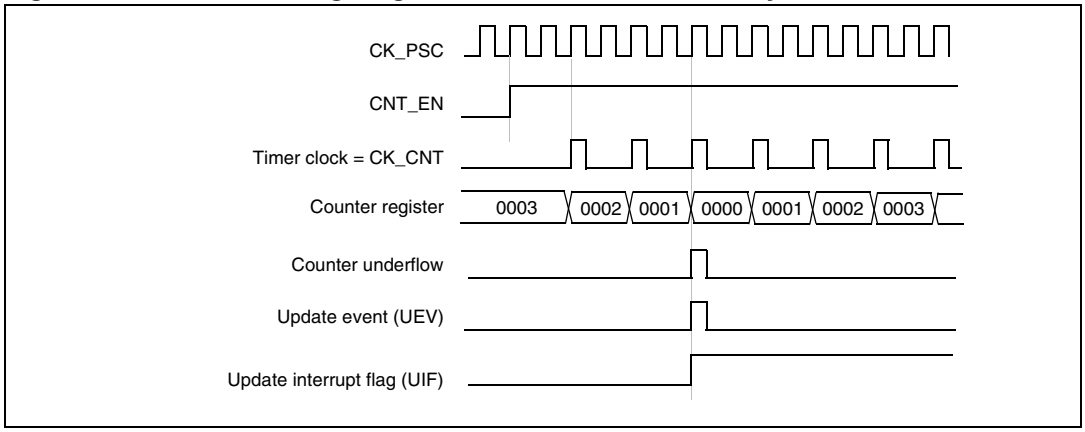


Figure 62. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36

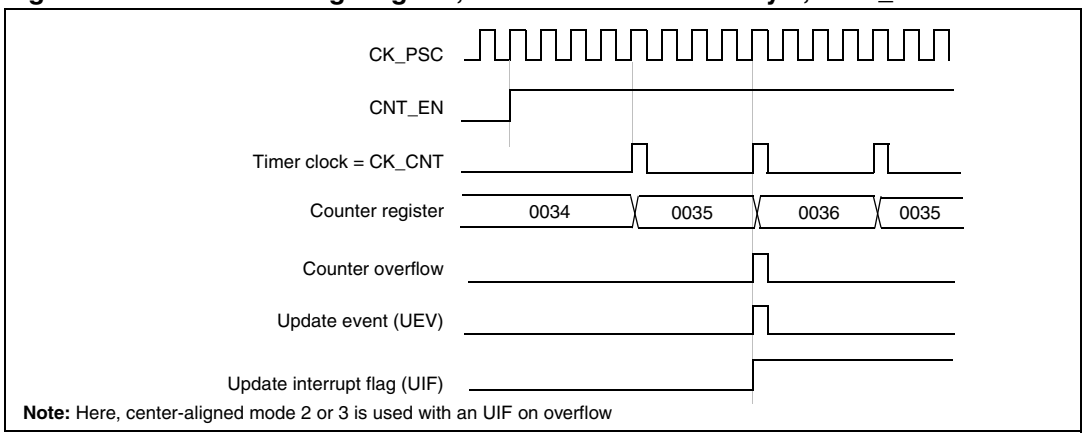


Figure 63. Counter timing diagram, internal clock divided by N

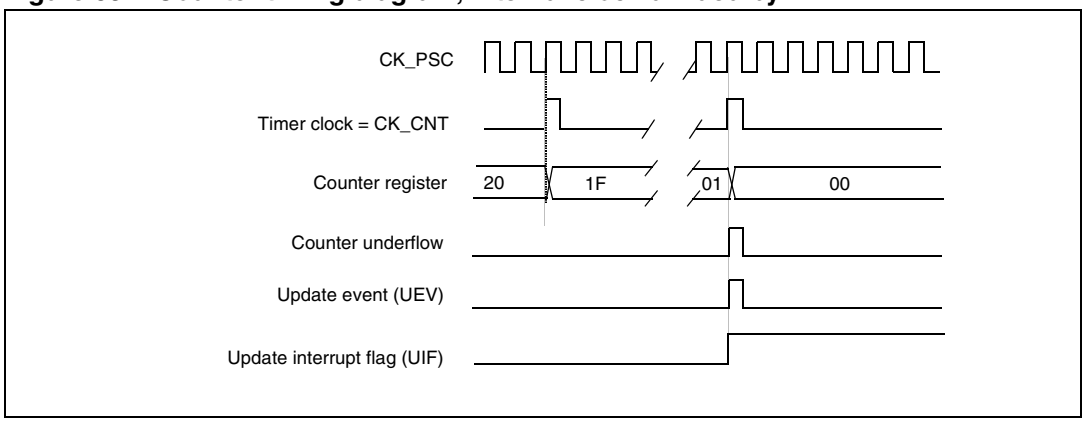


Figure 64. Counter timing diagram, update event with ARPE=1 (counter underflow)

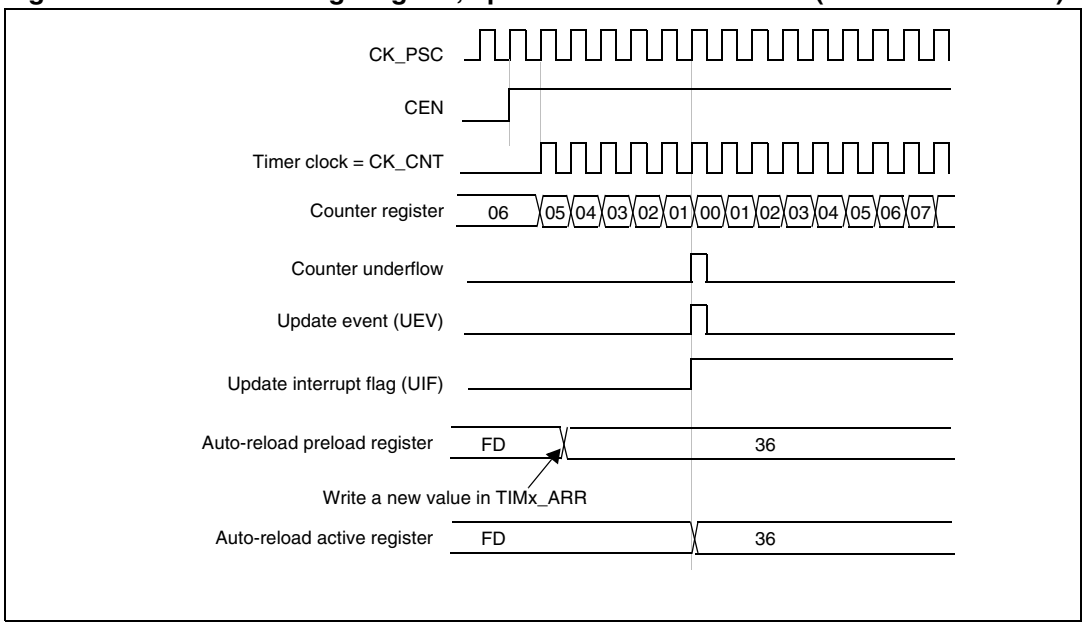
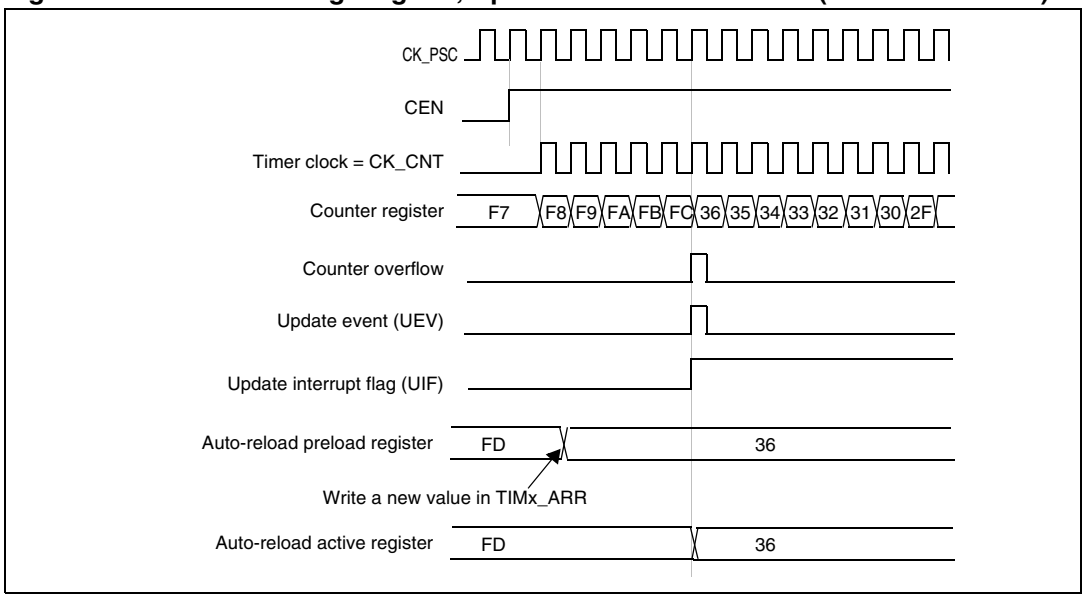


Figure 65. Counter timing diagram, Update event with ARPE=1 (counter overflow)



12.3.3 Repetition counter

Section 12.3.1: Time-base unit describes how the update event (UEV) is generated with respect to the counter overflows/underflows. It is actually generated only when the repetition counter has reached zero. This can be useful when generating PWM signals.

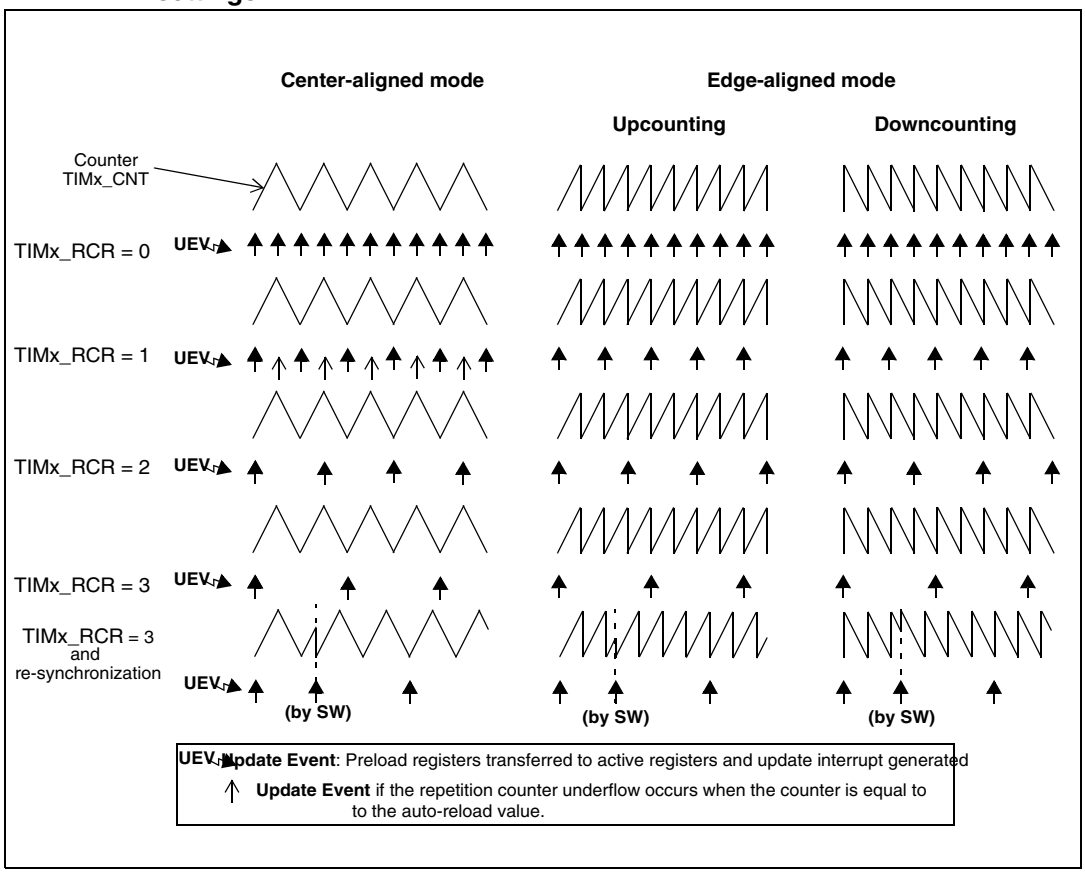
This means that data are transferred from the preload registers to the shadow registers (TIMx_ARR auto-reload register, TIMx_PSC prescaler register, but also TIMx_CCRx capture/compare registers in compare mode) every N counter overflows or underflows, where N is the value in the TIMx_RCR repetition counter register.

The repetition counter is decremented:

- At each counter overflow in upcounting mode,
- At each counter underflow in downcounting mode,
- At each counter overflow and at each counter underflow in center-aligned mode. Although this limits the maximum number of repetition to 128 PWM cycles, it makes it possible to update the duty cycle twice per PWM period. When refreshing compare registers only once per PWM period in center-aligned mode, maximum resolution is $2xT_{ck}$, due to the symmetry of the pattern.

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx_RCR register value (refer to *Figure 66*). When the update event is generated by software (by setting the UG bit in TIMx_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition counter is and the repetition counter is reloaded with the content of the TIMx_RCR register.

Figure 66. Update rate examples depending on mode and TIMx_RCR register settings



12.3.4 Clock selection

The counter clock can be provided by the following clock sources:

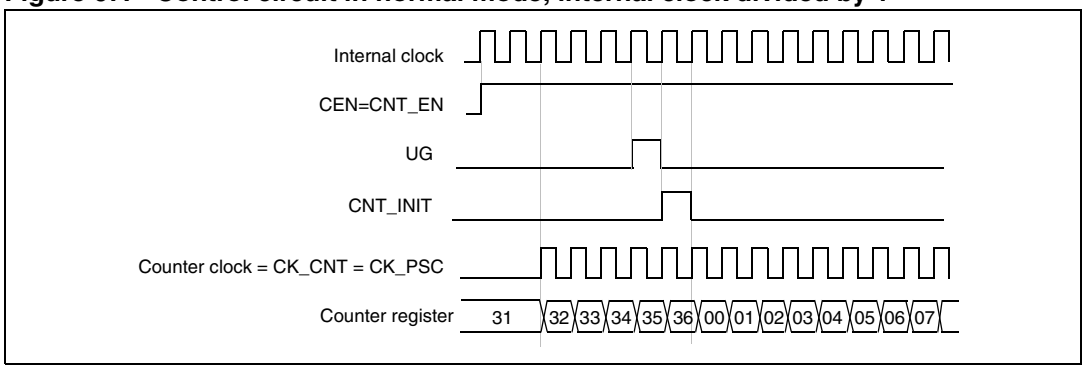
- Internal clock (CK_INT)
- External clock mode1: external input pin
- External clock mode2: external trigger input ETR
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to *Section : Using one timer as prescaler for another timer on page 300* for more details.

Internal clock source (CK_INT)

If the slave mode controller is disabled (SMS=000), then the CEN, DIR (in the TIMx_CR1 register) and UG bits (in the TIMx_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK_INT.

Figure 67 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

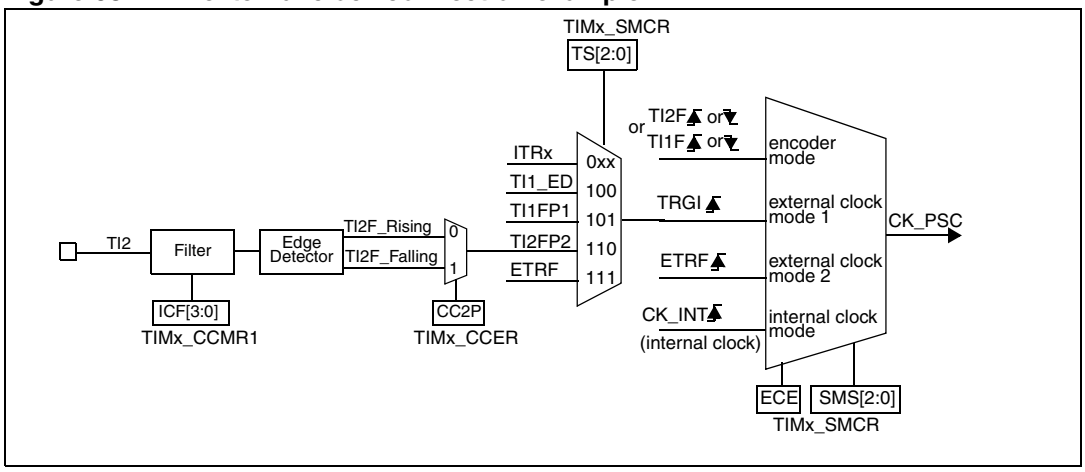
Figure 67. Control circuit in normal mode, internal clock divided by 1



External clock source mode 1

This mode is selected when SMS=111 in the TIMx_SMCR register. The counter can count at each rising or falling edge on a selected input.

Figure 68. TI2 external clock connection example



For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

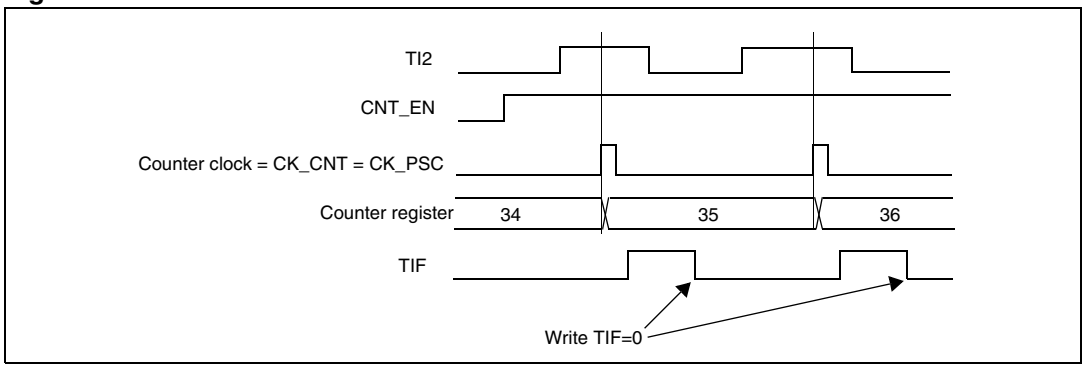
1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC2F=0000).
3. Select rising edge polarity by writing CC2P=0 in the TIMx_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx_SMCR register.
5. Select TI2 as the trigger input source by writing TS=110 in the TIMx_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

Note: The capture prescaler is not used for triggering, so you don't need to configure it.

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

The delay between the rising edge on TI2 and the actual clock of the counter is due to the resynchronization circuit on TI2 input.

Figure 69. Control circuit in external clock mode 1



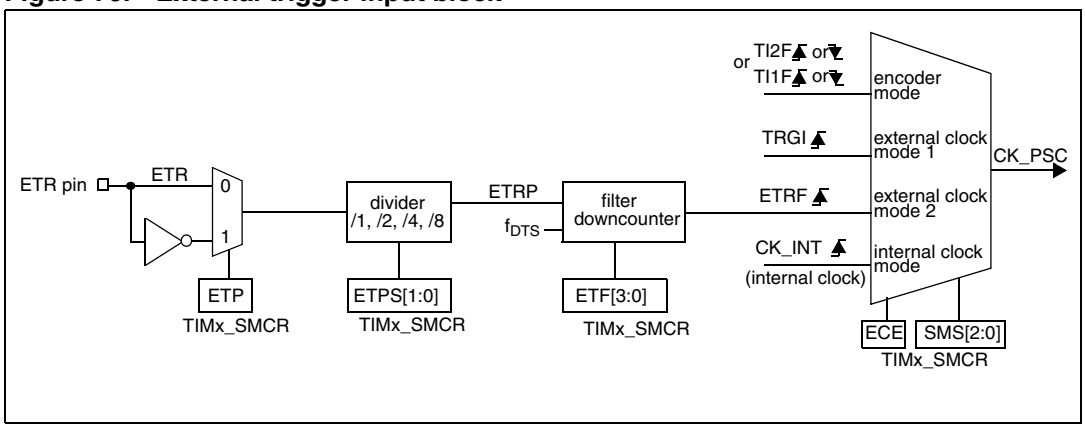
External clock source mode 2

This mode is selected by writing ECE=1 in the TIMx_SMCR register.

The counter can count at each rising or falling edge on the external trigger input ETR.

The *Figure 70* gives an overview of the external trigger input block.

Figure 70. External trigger input block



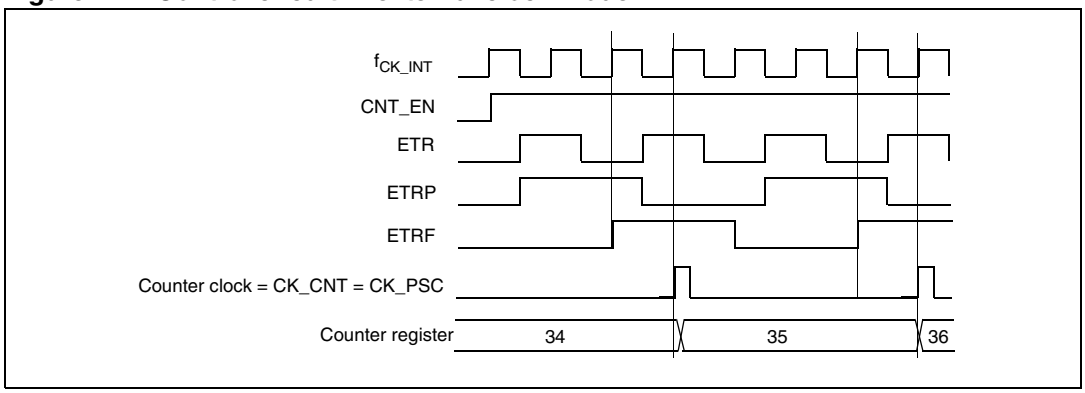
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0000 in the TIMx_SMCR register.
2. Set the prescaler by writing ETPS[1:0]=01 in the TIMx_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx_SMCR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR register.
5. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

The counter counts once each 2 ETR rising edges.

The delay between the rising edge on ETR and the actual clock of the counter is due to the resynchronization circuit on the ETRP signal.

Figure 71. Control circuit in external clock mode 2



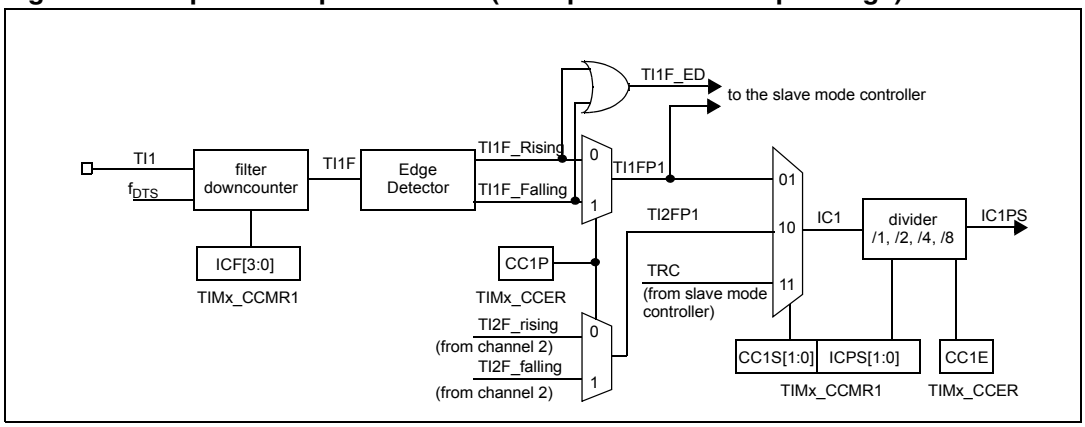
12.3.5 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

Figure 72 to *Figure 75* give an overview of one Capture/Compare channel.

The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

Figure 72. Capture/compare channel (example: channel 1 input stage)



The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 73. Capture/compare channel 1 main circuit

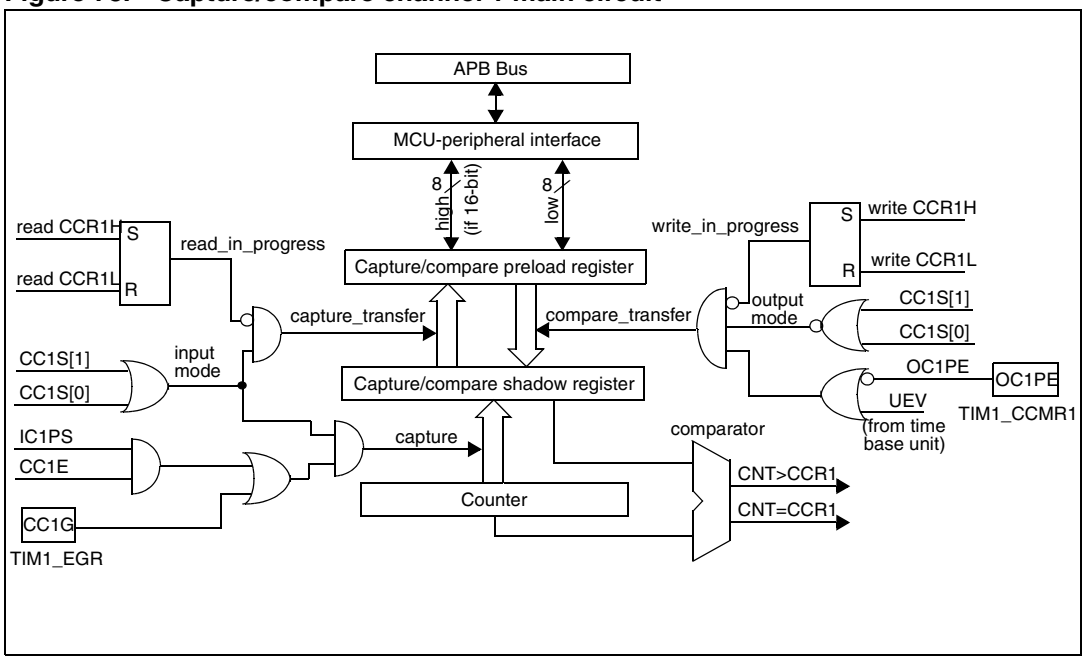


Figure 74. Output stage of capture/compare channel (channel 1 to 3)

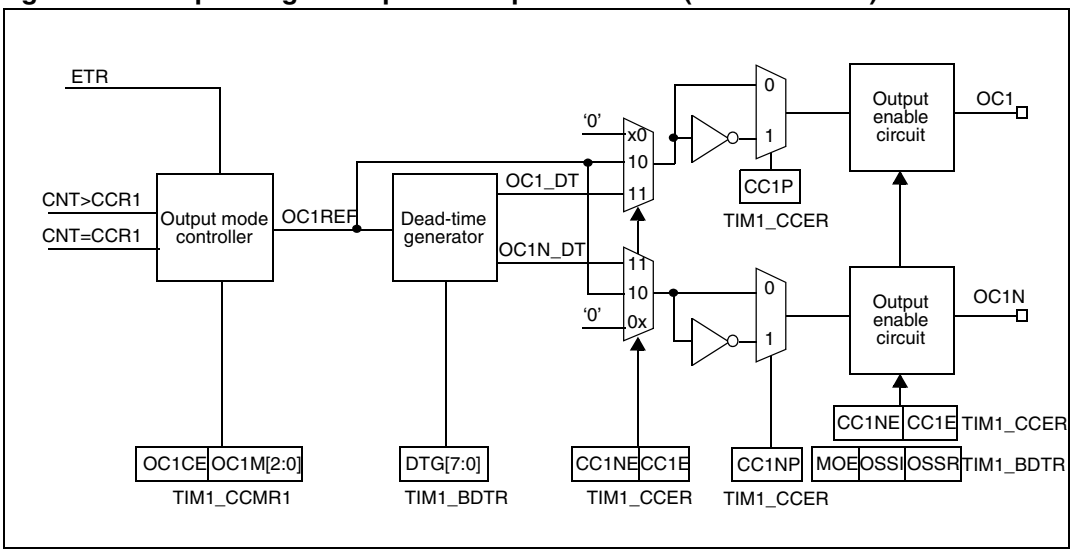
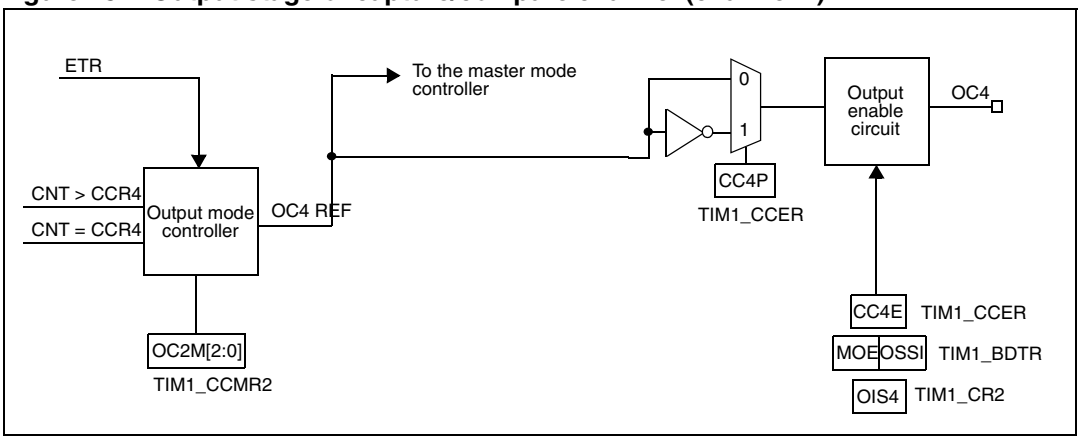


Figure 75. Output stage of capture/compare channel (channel 4)



The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

12.3.6 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCXIF flag (TIMx_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx_CCRx register. CCxOF is cleared when you write it to '0'.

The following example shows how to capture the counter value in TIMx_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx_CCR1 register becomes read-only.
- Program the input filter duration you need with respect to the signal you connect to the timer (when the input is one of the TIx (ICxF bits in the TIMx_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at must 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been detected (sampled at f_{DTS} frequency). Then write IC1F bits to 0011 in the TIMx_CCMR1 register.
- Select the edge of the active transition on the TI1 channel by writing CC1P bit to 0 in the TIMx_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx_DIER register.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

Note: IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx_EGR register.

12.3.7 PWM input mode

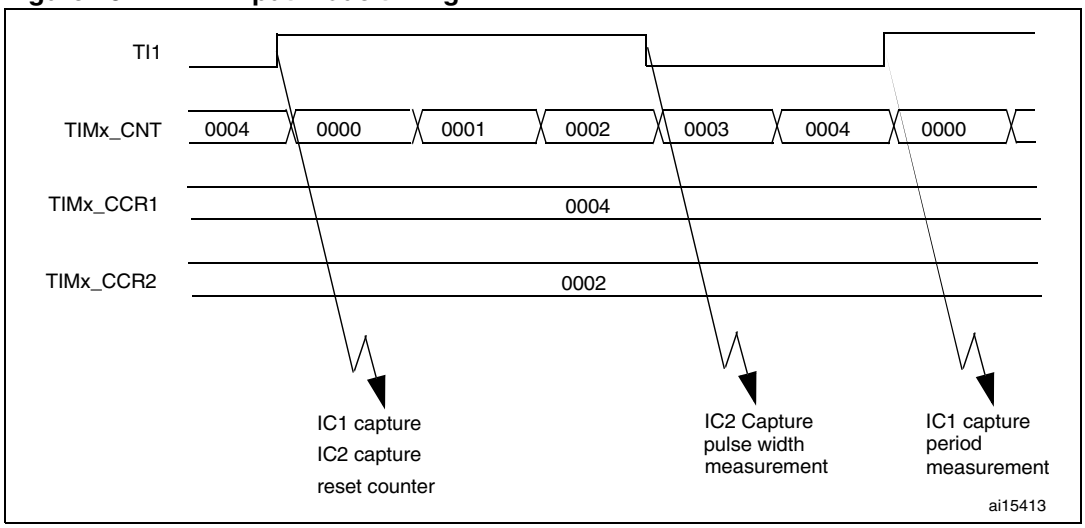
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, you can measure the period (in TIMx_CCR1 register) and the duty cycle (in TIMx_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK_INT frequency and prescaler value):

- Select the active input for TIMx_CCR1: write the CC1S bits to 01 in the TIMx_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): write the CC1P bit to '0' (active on rising edge).
- Select the active input for TIMx_CCR2: write the CC2S bits to 10 in the TIMx_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P bit to '1' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER register.

Figure 76. PWM input mode timing



1. The PWM input mode can be used only with the TIMx_CH1/TIMx_CH2 signals due to the fact that only TI1FP1 and TI2FP2 are connected to the slave mode controller.

12.3.8 Forced output mode

In output mode (CCxS bits = 00 in the TIMx_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, you just need to write 101 in the OCxM bits in the corresponding TIMx_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP=0 (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to 100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

12.3.9 Output compare mode

This function is used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the capture/compare register and the counter, the output compare function:

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx_CCMRx register) and the output polarity (CCxP bit in the TIMx_CCER register). The output pin can keep its level (OCXM=000), be set active (OCxM=001), be set inactive (OCxM=010) or can toggle (OCxM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx_DIER register, CCDS bit in the TIMx_CR2 register for the DMA request selection).

The TIMx_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx_CCMRx register.

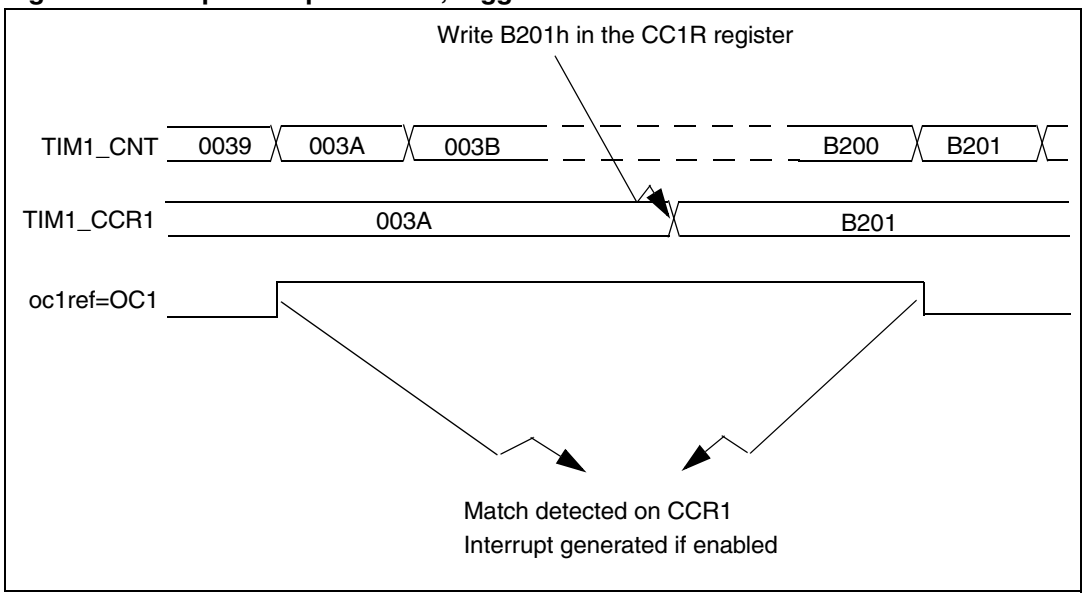
In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One Pulse mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
   - Write OCxM = 011 to toggle OCx output pin when CNT matches CCRx
   - Write OCxPE = 0 to disable preload register
   - Write CCxP = 0 to select active high polarity
   - Write CCxE = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx_CR1 register.

The TIMx_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx_CCRx shadow register is updated only at the next update event UEV). An example is given in *Figure 77*.

Figure 77. Output compare mode, toggle on OC1.



12.3.10 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '110' (PWM mode 1) or '111' (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by a combination of the CCxE, CCxNE, MOE, OSSI and OSSR bits (TIMx_CCER and TIMx_BDTR registers). Refer to the TIMx_CCER register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx ≤TIMx_CNT or TIMx_CNT ≤TIMx_CCRx (depending on the direction of the counter).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

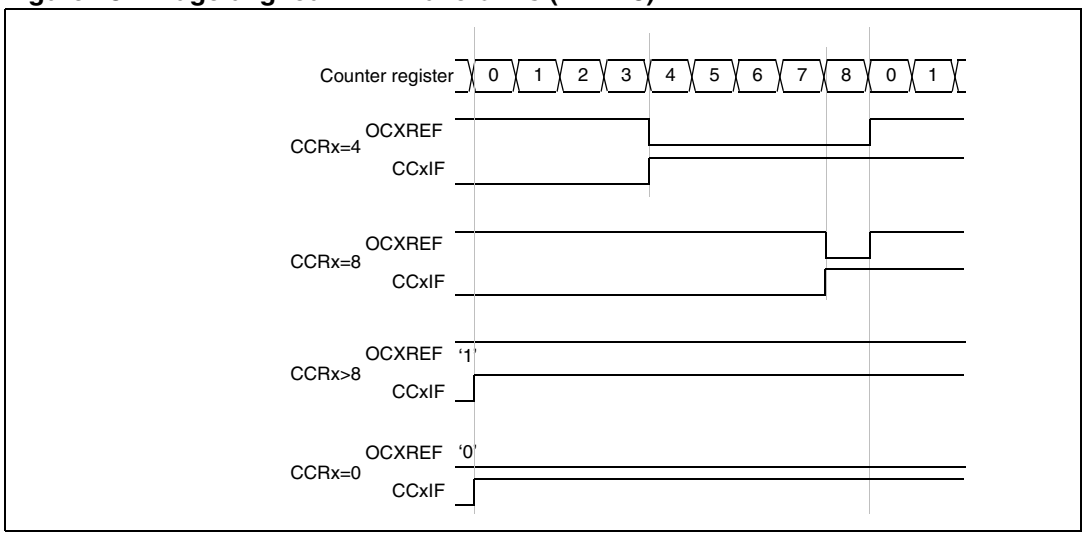
PWM edge-aligned mode

- Upcounting configuration

  Upcounting is active when the DIR bit in the TIMx_CR1 register is low. Refer to the *Upcounting mode on page 203*.

  In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as TIMx_CNT < TIMx_CCRx else it becomes low. If the compare value in TIMx_CCRx is greater than the auto-reload value (in TIMx_ARR) then OCxREF is held at '1'. If the compare value is 0 then OCxRef is held at '0'. *Figure 78* shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

Figure 78. Edge-aligned PWM waveforms (ARR=8)



- Downcounting configuration

  Downcounting is active when DIR bit in TIMx_CR1 register is high. Refer to the *Downcounting mode on page 206*

  In PWM mode 1, the reference signal OCxRef is low as long as TIMx_CNT > TIMx_CCRx else it becomes high. If the compare value in TIMx_CCRx is greater than the auto-reload value in TIMx_ARR, then OCxREF is held at '1'. 0% PWM is not possible in this mode.

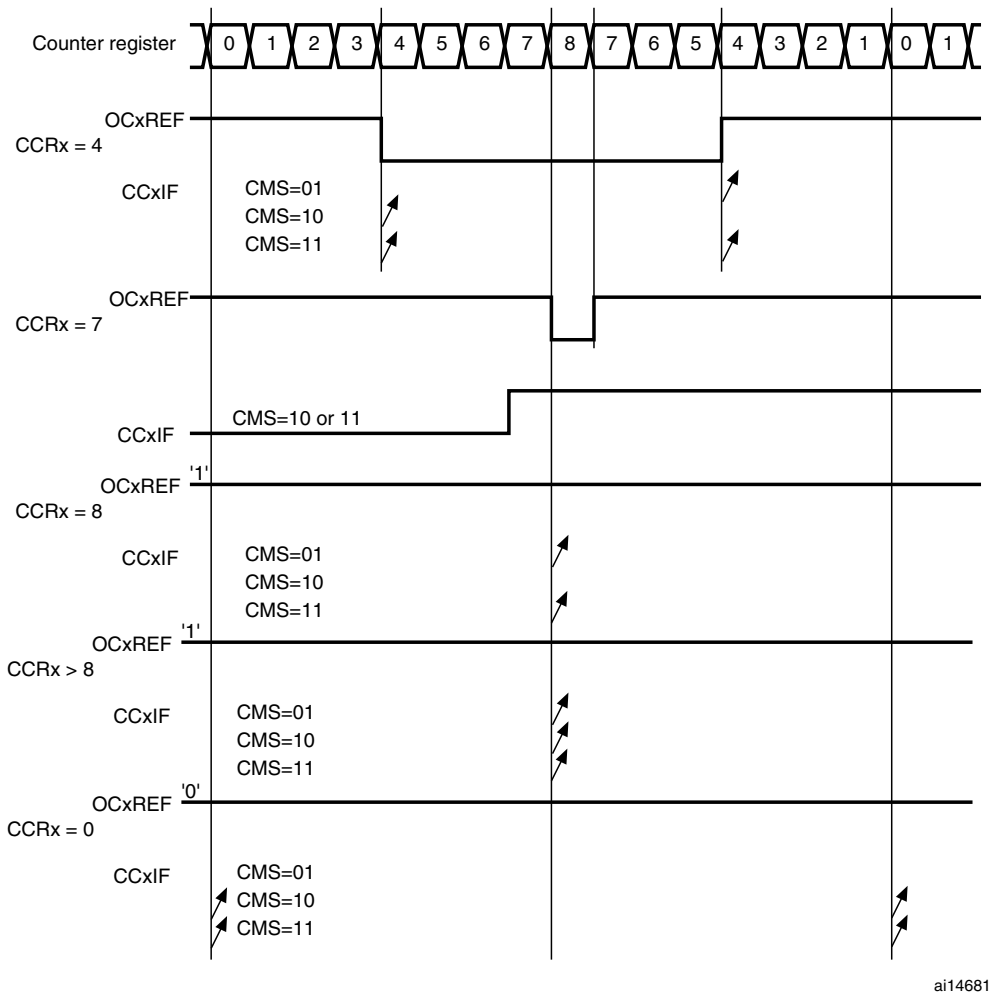
PWM center-aligned mode

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are different from '00' (all the remaining configurations having the same effect on the OCxRef/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx_CR1 register is updated by hardware and must not be changed by software. Refer to the *Center-aligned mode (up/down counting) on page 208*.

Figure 79 shows some center-aligned PWM waveforms in an example where:

- TIMx_ARR=8,
- PWM mode is the PWM mode 1,
- The flag is set when the counter counts down corresponding to the center-aligned mode 1 selected for CMS=01 in TIMx_CR1 register.

Figure 79. Center-aligned PWM waveforms (ARR=8)



Hints on using center-aligned mode:

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter counts up or down depending on the value written in the DIR bit in the TIMx_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if you write a value in the counter that is greater than the auto-reload value (TIMx_CNT>TIMx_ARR). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if you write 0 or write the TIMx_ARR value in the counter but no Update Event UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIMx_EGR register) just before starting the counter and not to write the counter while it is running.

12.3.11 Complementary outputs and dead-time insertion

The advanced-control timers (TIM1&TIM8) can output two complementary signals and manage the switching-off and the switching-on instants of the outputs.

This time is generally known as dead-time and you have to adjust it depending on the devices you have connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

You can select the polarity of the outputs (main output OCx or complementary OCxN) independently for each output. This is done by writing to the CCxP and CCxNP bits in the TIMx_CCER register.

The complementary signals OCx and OCxN are activated by a combination of several control bits: the CCxE and CCxNE bits in the TIMx_CCER register and the MOE, OISx, OISxN, OSSI and OSSR bits in the TIMx_BDTR and TIMx_CR2 registers. Refer to *Table 56: Output control bits for complementary OCx and OCxN channels with break feature on page 256* for more details. In particular, the dead-time is activated when switching to the IDLE state (MOE falling down to 0).

Dead-time insertion is enabled by setting both CCxE and CCxNE bits, and the MOE bit if the break circuit is present. There is one 10-bit dead-time generator for each channel. From a reference waveform OCxREF, it generates 2 outputs OCx and OCxN. If OCx and OCxN are active high:

- The OCx output signal is the same as the reference signal except for the rising edge, which is delayed relative to the reference rising edge.
- The OCxN output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference falling edge.

If the delay is greater than the width of the active output (OCx or OCxN) then the corresponding pulse is not generated.

The following figures show the relationships between the output signals of the dead-time generator and the reference signal OCxREF. (we suppose CCxP=0, CCxNP=0, MOE=1, CCxE=1 and CCxNE=1 in these examples)

Figure 80. Complementary output with dead-time insertion.

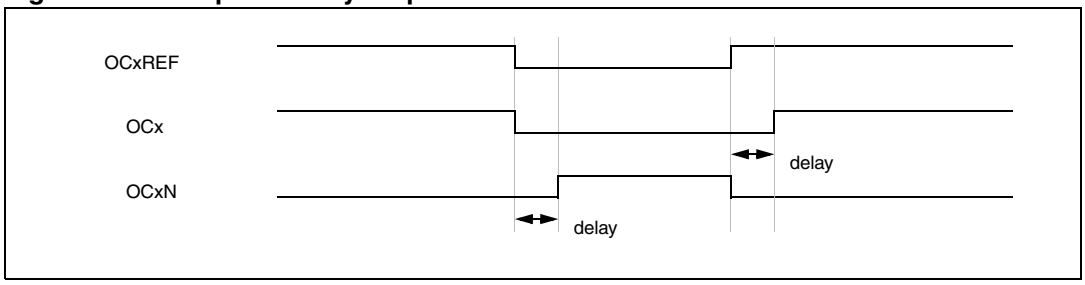


Figure 81. Dead-time waveforms with delay greater than the negative pulse.

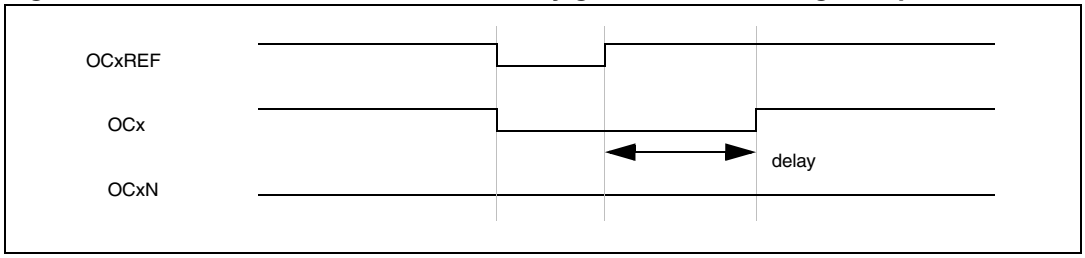
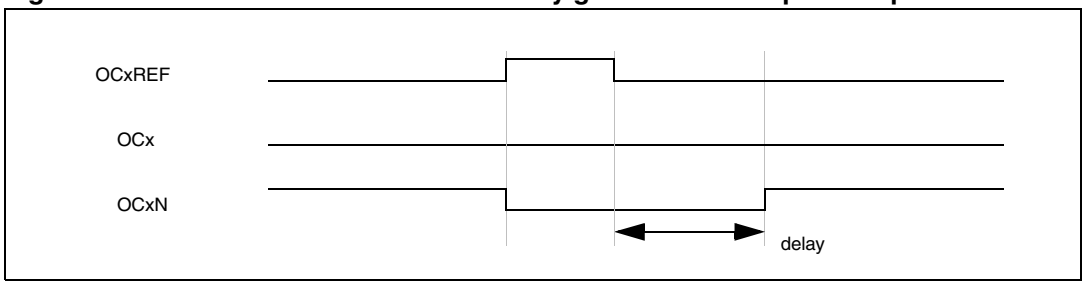


Figure 82. Dead-time waveforms with delay greater than the positive pulse.



The dead-time delay is the same for each of the channels and is programmable with the DTG bits in the TIMx_BDTR register. Refer to *Section 12.4.18: Break and dead-time register (TIMx_BDTR) on page 260* for delay calculation.

Re-directing OCxREF to OCx or OCxN

In output mode (forced, output compare or PWM), OCxREF can be re-directed to the OCx output or to OCxN output by configuring the CCxE and CCxNE bits in the TIMx_CCER register.

This allows you to send a specific waveform (such as PWM or static active level) on one output while the complementary remains at its inactive level. Other alternative possibilities are to have both outputs at inactive level or both outputs active and complementary with dead-time.

Note: When only OCxN is enabled (CCxE=0, CCxNE=1), it is not complemented and becomes active as soon as OCxREF is high. For example, if CCxNP=0 then OCxN=OCxRef. On the other hand, when both OCx and OCxN are enabled (CCxE=CCxNE=1) OCx becomes active when OCxREF is high whereas OCxN is complemented and becomes active when OCxREF is low.

12.3.12 Using the break function

When using the break function, the output enable signals and inactive levels are modified according to additional control bits (MOE, OSSI and OSSR bits in the TIMx_BDTR register, OISx and OISxN bits in the TIMx_CR2 register). In any case, the OCx and OCxN outputs cannot be set both to active level at a given time. Refer to *Table 56: Output control bits for complementary OCx and OCxN channels with break feature on page 256* for more details.

The break source can be either the break input pin or a clock failure event, generated by the Clock Security System (CSS), from the Reset Clock Controller. For further information on the Clock Security System, refer to *Section 6.2.7: Clock security system (CSS) on page 72*.

When exiting from reset, the break circuit is disabled and the MOE bit is low. You can enable the break function by setting the BKE bit in the TIMx_BDTR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if you write MOE to 1 whereas it was low, you must insert a delay (dummy instruction) before reading it correctly. This is because you write the asynchronous signal and read the synchronous signal.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or in reset state (selected by the OSSI bit). This feature functions even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OISx bit in the TIMx_CR2 register as soon as MOE=0. If OSSI=0 then the timer releases the enable output else the enable output remains high.
- When complementary outputs are used:
  - The outputs are first put in reset state inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
  - If the timer clock is still present, then the dead-time generator is reactivated in order to drive the outputs with the level programmed in the OISx and OISxN bits after a dead-time. Even in this case, OCx and OCxN cannot be driven to their active level together. Note that because of the resynchronization on MOE, the dead-time duration is a bit longer than usual (around 2 ck_tim clock cycles).
  - If OSSI=0 then the timer releases the enable outputs else the enable outputs remain or become high as soon as one of the CCxE or CCxNE bits is high.
- The break status flag (BIF bit in the TIMx_SR register) is set. An interrupt can be generated if the BIE bit in the TIMx_DIER register is set. A DMA request can be sent if the BDE bit in the TIMx_DIER register is set.
- If the AOE bit in the TIMx_BDTR register is set, the MOE bit is automatically set again at the next update event UEV. This can be used to perform a regulation, for instance. Else, MOE remains low until you write it to ‘1’ again. In this case, it can be used for security and you can connect the break input to an alarm from power drivers, thermal sensors or any security components.

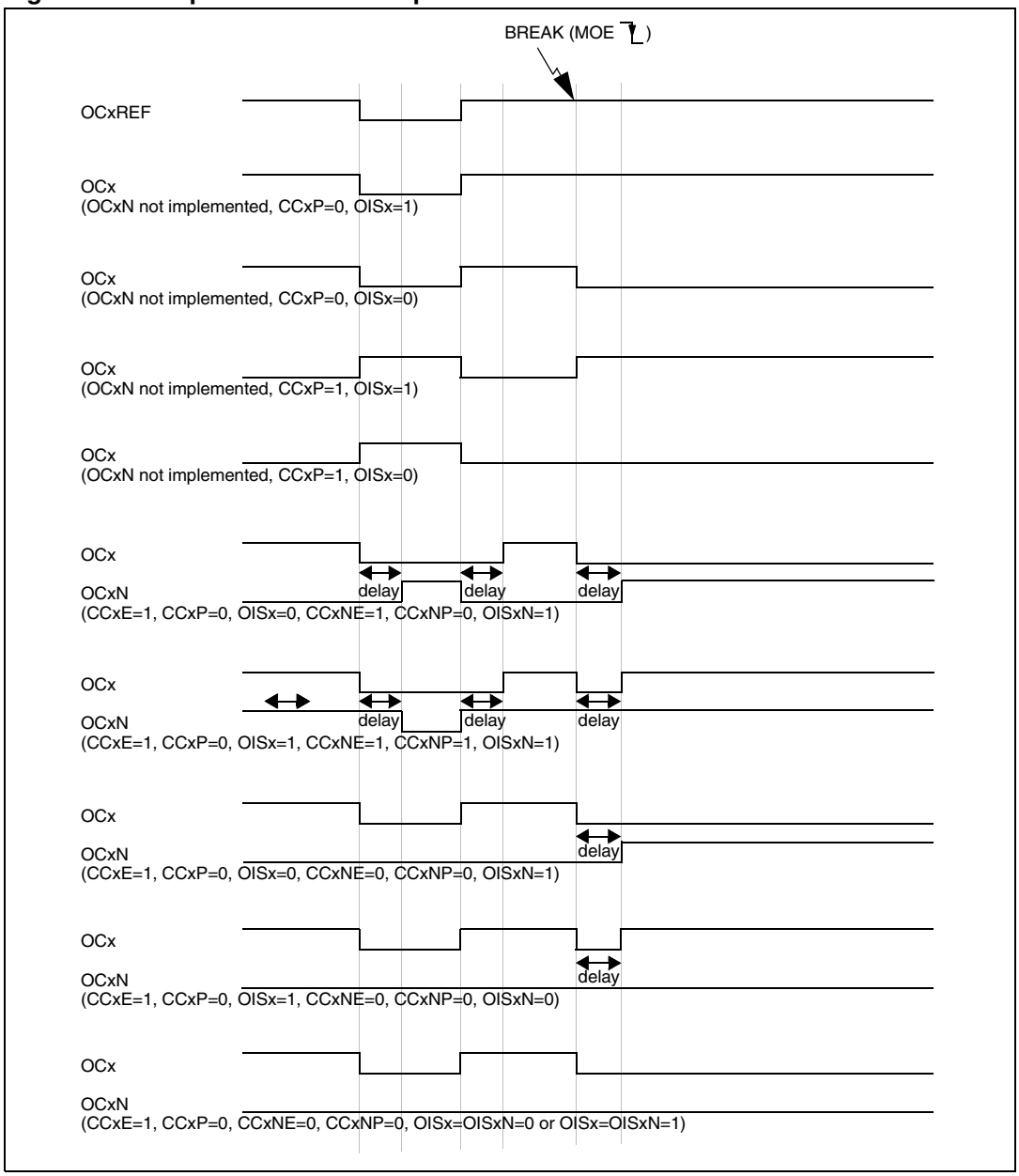
Note: The break inputs is acting on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.

The break can be generated by the BRK input which has a programmable polarity and an enable bit BKE in the TIMx_BDTR Register.

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows you to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). You can choose from 3 levels of protection selected by the LOCK bits in the TIMx_BDTR register. Refer to *Section 12.4.18: Break and dead-time register (TIMx_BDTR) on page 260*. The LOCK bits can be written only once after an MCU reset.

The *Figure 83* shows an example of behavior of the outputs in response to a break.

Figure 83. Output behavior in response to a break.



12.3.13 Clearing the OCxREF signal on an external event

The OCxREF signal for a given channel can be driven Low by applying a High level to the ETRF input (OCxCE enable bit of the corresponding TIMx_CCMRx register set to ‘1’). The OCxREF signal remains Low until the next update event, UEV, occurs.

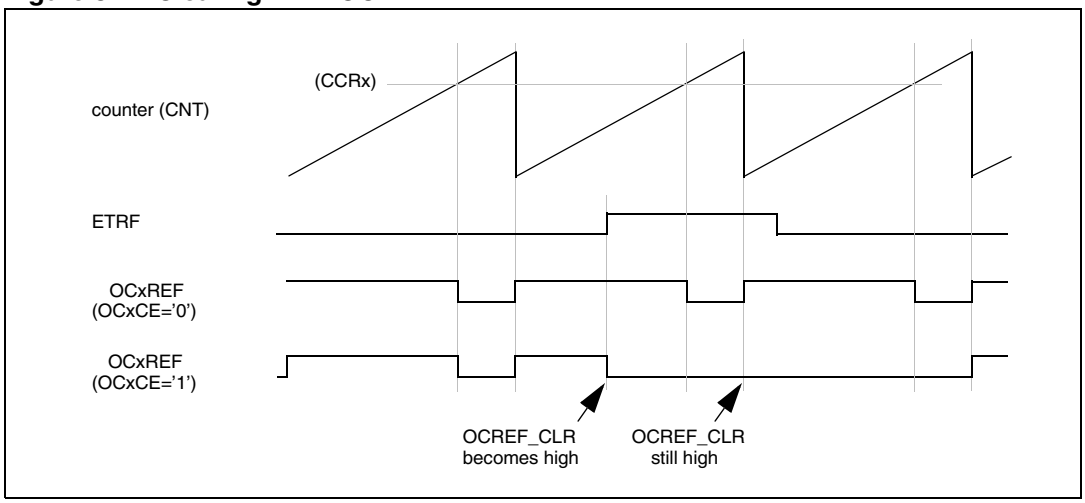
This function can only be used in output compare and PWM modes, and does not work in forced mode.

For example, the OCxREF signal) can be connected to the output of a comparator to be used for current handling. In this case, the ETR must be configured as follow:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] of the TIMx_SMCR register set to ‘00’.
2. The external clock mode 2 must be disabled: bit ECE of the TIMx_SMCR register set to ‘0’.
3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured according to the user needs.

Figure 84 shows the behavior of the OCxREF signal when the ETRF Input becomes High, for both values of the enable bit OCxCE. In this example, the timer TIMx is programmed in PWM mode.

Figure 84. Clearing TIMx OCxREF



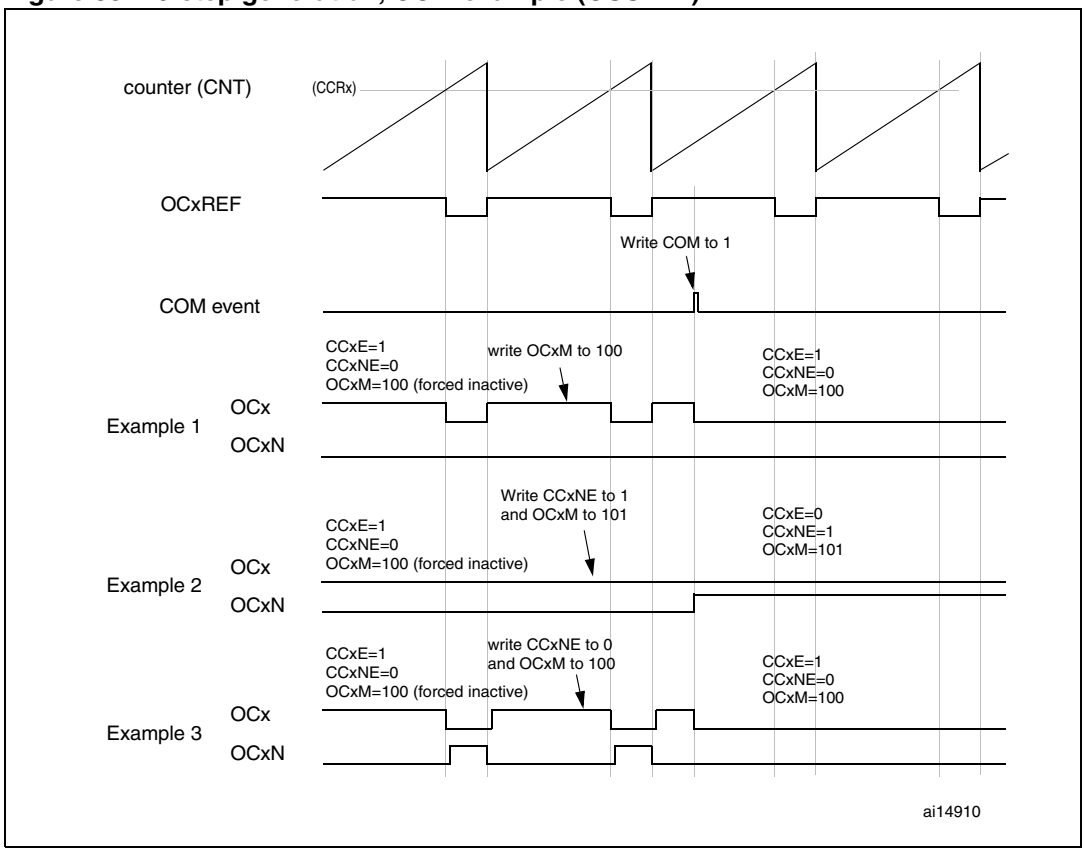
12.3.14 6-step PWM generation

When complementary outputs are used on a channel, preload bits are available on the OCxM, CCxE and CCxNE bits. The preload bits are transferred to the shadow bits at the COM commutation event. Thus you can program in advance the configuration for the next step and change the configuration of all the channels at the same time. COM can be generated by software by setting the COM bit in the TIMx_EGR register or by hardware (on TRGI rising edge).

A flag is set when the COM event occurs (COMIF bit in the TIMx_SR register), which can generate an interrupt (if the COMIE bit is set in the TIMx_DIER register) or a DMA request (if the COMDE bit is set in the TIMx_DIER register).

The *Figure 85* describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

Figure 85. 6-step generation, COM example (OSSR=1)



12.3.15 One-pulse mode

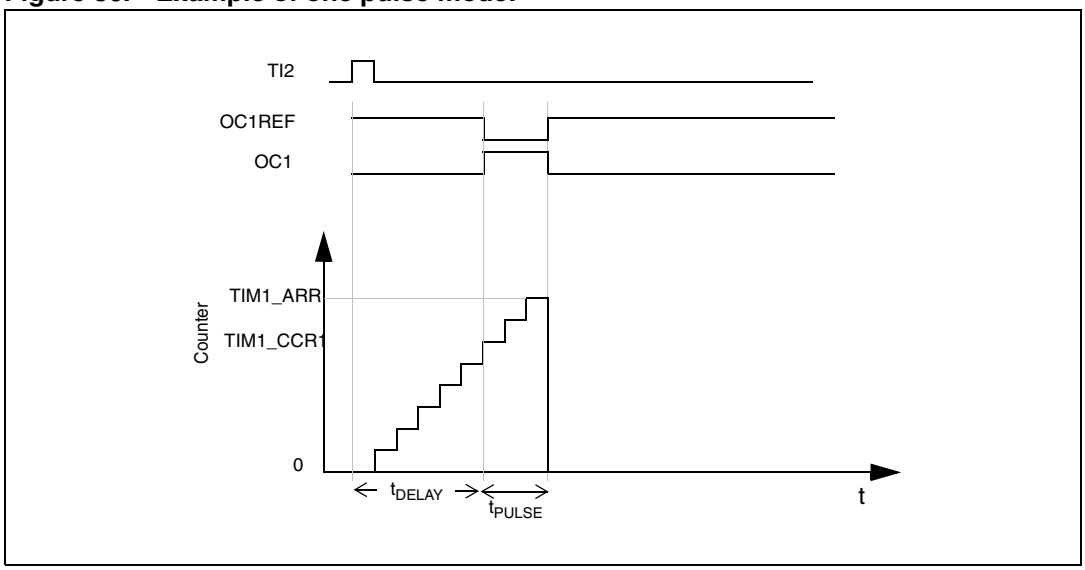
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. You select One-pulse mode by setting the OPM bit in the TIMx_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

- In upcounting: CNT < CCRx ≤ ARR (in particular, 0 < CCRx)
- In downcounting: CNT > CCRx

Figure 86. Example of one pulse mode.



For example you may want to generate a positive pulse on OC1 with a length of t_{PULSE} and after a delay of t_{DELAY} as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 to TI2 by writing CC2S='01' in the TIMx_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' in the TIMx_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIMx_SMCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

- The t_{DELAY} is defined by the value written in the TIMx_CCR1 register.
- The t_{PULSE} is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1).
- Let's say you want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this you enable PWM mode 2 by writing OC1M=111 in the TIMx_CCMR1 register. You can optionally enable the preload registers by writing OC1PE='1' in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case you have to write the compare value in the TIMx_CCR1 register, the auto-reload value in the TIMx_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx_CR1 register should be low.

You only want 1 pulse, so you write '1' in the OPM bit in the TIMx_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

Particular case: OCx fast enable:

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay t_{DELAY} min we can get.

If you want to output a waveform with the minimum delay, you can set the OCxFE bit in the TIMx_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

12.3.16 Encoder interface mode

To select Encoder Interface mode write SMS='001' in the TIMx_SMCR register if the counter is counting on TI2 edges only, SMS='010' if it is counting on TI1 edges only and SMS='011' if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx_CCER register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to *Table 54*. The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIMx_ARR before starting. in the same way, the capture, compare, prescaler,

repetition counter, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

Table 54. Counting direction versus encoder signals

Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No Count	No Count
	Low	Up	Down	No Count	No Count
Counting on TI2 only	High	No Count	No Count	Up	Down
	Low	No Count	No Count	Down	Up
Counting on TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators are normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

The *Figure 87* gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S='01' (TIMx_CCMR1 register, TI1FP1 mapped on TI1).
- CC2S='01' (TIMx_CCMR2 register, TI1FP2 mapped on TI2).
- CC1P='0' (TIMx_CCER register, TI1FP1 non-inverted, TI1FP1=TI1).
- CC2P='0' (TIMx_CCER register, TI1FP2 non-inverted, TI1FP2= TI2).
- SMS='011' (TIMx_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIMx_CR1 register, Counter enabled).

Figure 87. Example of counter operation in encoder interface mode.

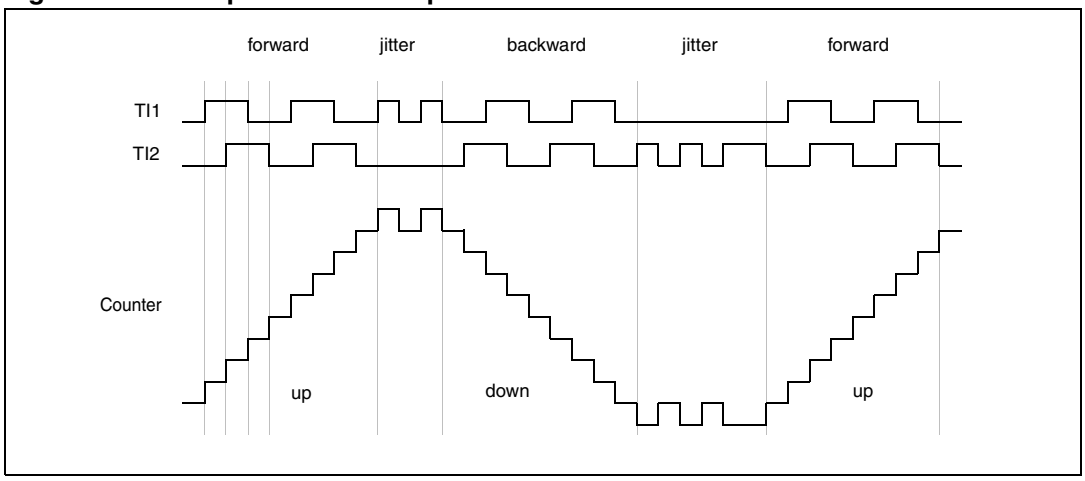
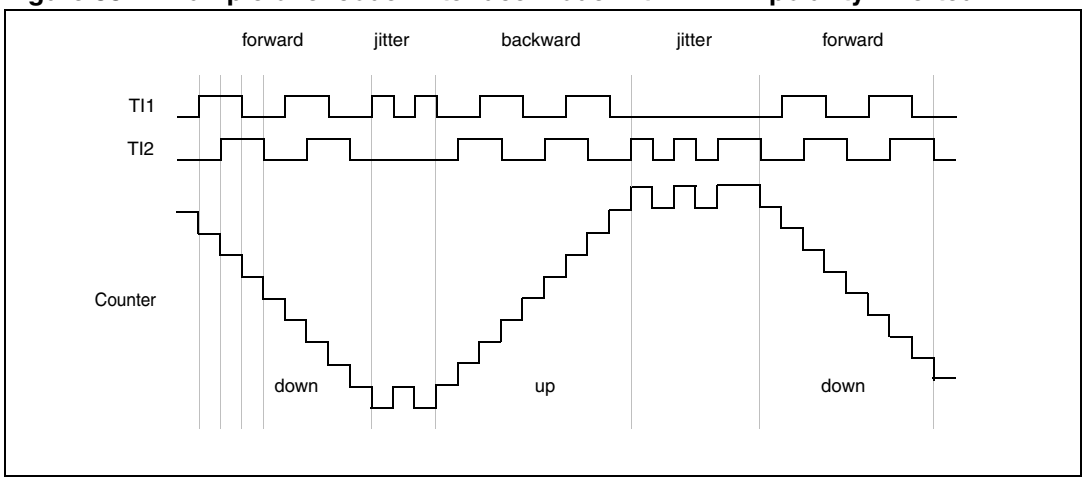


Figure 88 gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P='1').

Figure 88. Example of encoder interface mode with TI1FP1 polarity inverted.



The timer, when configured in Encoder Interface mode provides information on the sensor's current position. You can obtain dynamic information (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. You can do this by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). when available, it is also possible to read its value through a DMA request generated by a real-time clock.

12.3.17 Timer input XOR function

The TI1S bit in the TIMx_CR2 register, allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx_CH1, TIMx_CH2 and TIMx_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture. An example of this feature used to interface Hall sensors is given in *Section 12.3.18* below.

12.3.18 Interfacing with Hall sensors

This is done using the advanced-control timers (TIM1 or TIM8) to generate PWM signals to drive the motor and another timer TIMx (TIM2, TIM3, TIM4 or TIM5) referred to as "interfacing timer" in *Figure 89*. The "interfacing timer" captures the 3 timer input pins (CC1, CC2, CC3) connected through a XOR to the TI1 input channel (selected by setting the TI1S bit in the TIMx_CR2 register).

The slave mode controller is configured in reset mode; the slave input is TI1F_ED. Thus, each time one of the 3 inputs toggles, the counter restarts counting from 0. This creates a time base triggered by any change on the Hall inputs.

On the "interfacing timer", capture/compare channel 1 is configured in capture mode, capture signal is TRC (See *Figure 72: Capture/compare channel (example: channel 1 input stage) on page 216*). The captured value, which corresponds to the time elapsed between 2 changes on the inputs, gives information about motor speed.

The "interfacing timer" can be used in output mode to generate a pulse which changes the configuration of the channels of the advanced-control timer (TIM1 or TIM8) (by triggering a COM event). The TIM1 timer is used to generate PWM signals to drive the motor. To do this, the interfacing timer channel must be programmed so that a positive pulse is generated after a programmed delay (in output compare or PWM mode). This pulse is sent to the advanced-control timer (TIM1 or TIM8) through the TRGO output.

Example: you want to change the PWM configuration of your advanced-control timer TIM1 after a programmed delay each time a change occurs on the Hall inputs connected to one of the TIMx timers.

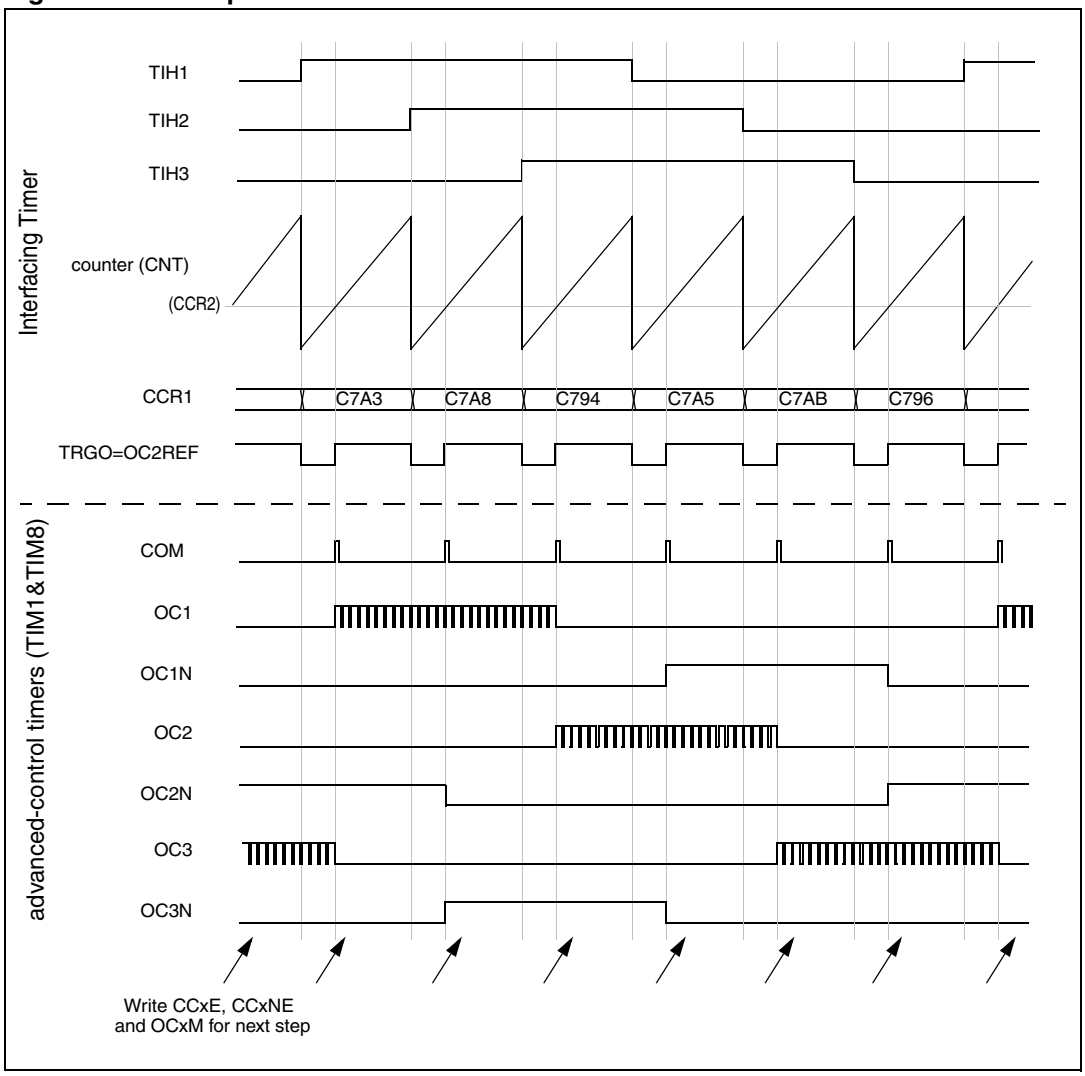
- Configure 3 timer inputs ORed to the TI1 input channel by writing the TI1S bit in the TIMx_CR2 register to '1',
- Program the time base: write the TIMx_ARR to the max value (the counter must be cleared by the TI1 change. Set the prescaler to get a maximum counter period longer than the time between 2 changes on the sensors,
- Program the channel 1 in capture mode (TRC selected): write the CC1S bits in the TIMx_CCMR1 register to '01'. You can also program the digital filter if needed,
- Program the channel 2 in PWM 2 mode with the desired delay: write the OC2M bits to '111' and the CC2S bits to '00' in the TIMx_CCMR1 register,
- Select OC2REF as trigger output on TRGO: write the MMS bits in the TIMx_CR2 register to '101',

In the advanced-control timer TIM1, the right ITR input must be selected as trigger input, the timer is programmed to generate PWM signals, the capture/compare control signals are preloaded (CCPC=1 in the TIMx_CR2 register) and the COM event is controlled by the trigger input (CCUS=1 in the TIMx_CR2 register). The PWM control bits (CCxE, OCxM) are

written after a COM event for the next step (this can be done in an interrupt subroutine generated by the rising edge of OC2REF).

The *Figure 89* describes this example.

Figure 89. Example of hall sensor interface



12.3.19 TIMx and external trigger synchronization

The TIMx timer can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx_ARR, TIMx_CCRx) are updated.

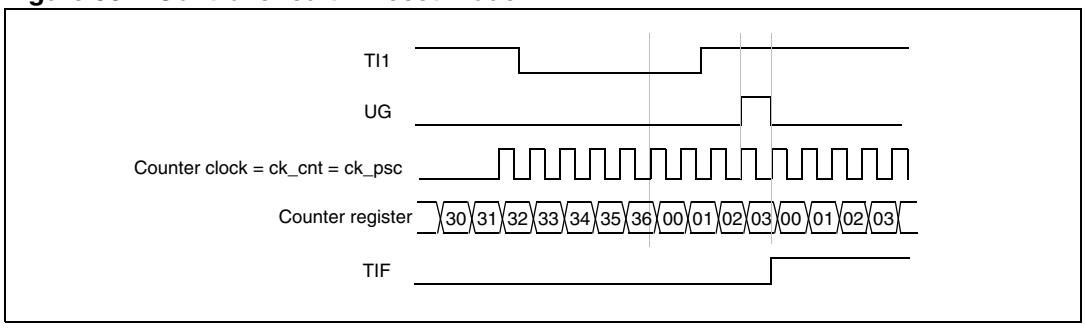
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx_CCMR1 register. Write CC1P=0 in TIMx_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Start the counter by writing CEN=1 in the TIMx_CR1 register.

The counter starts counting on the internal clock, then behaves normally until TI1 rising edge. When TI1 rises, the counter is cleared and restarts from 0. In the meantime, the trigger flag is set (TIF bit in the TIMx_SR register) and an interrupt request, or a DMA request can be sent if enabled (depending on the TIE and TDE bits in TIMx_DIER register).

The following figure shows this behavior when the auto-reload register TIMx_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

Figure 90. Control circuit in reset mode



Slave mode: Gated mode

The counter can be enabled depending on the level of a selected input.

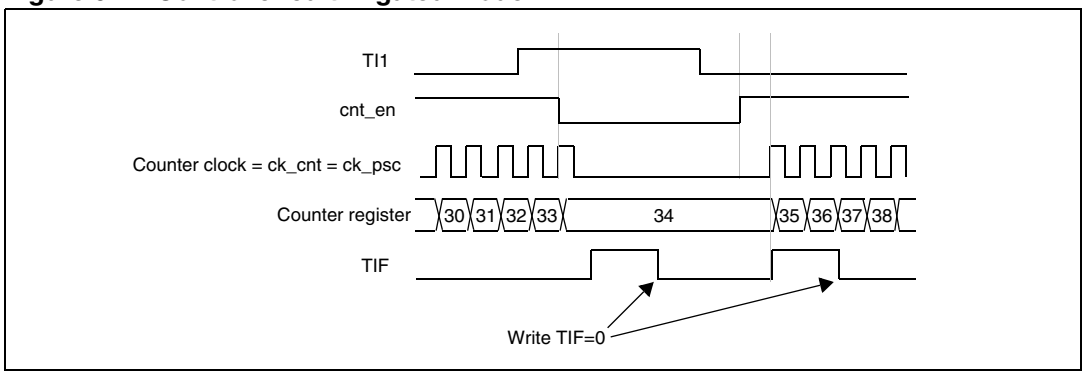
In the following example, the upcounter counts only when TI1 input is low:

- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S=01 in TIMx_CCMR1 register. Write CC1P=1 in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

The counter starts counting on the internal clock as long as TI1 is low and stops as soon as TI1 becomes high. The TIF flag in the TIMx_SR register is set both when the counter starts or stops.

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

Figure 91. Control circuit in gated mode



Slave mode: Trigger mode

The counter can start in response to an event on a selected input.

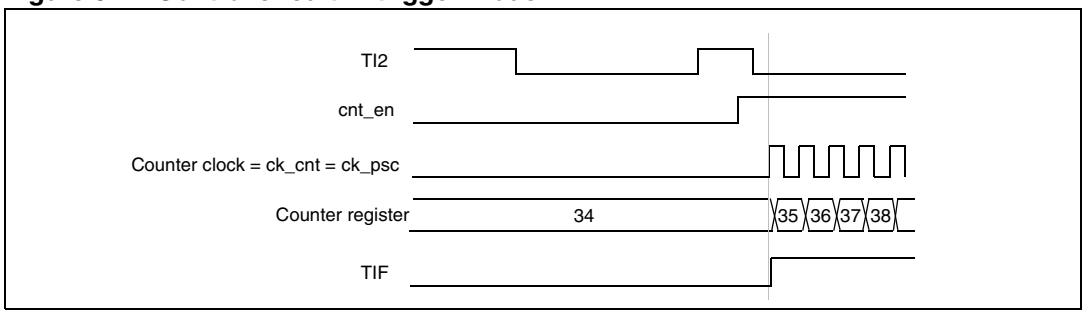
In the following example, the upcounter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC2S bits are configured to select the input capture source only, CC2S=01 in TIMx_CCMR1 register. Write CC2P=1 in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx_SMCR register.

When a rising edge occurs on TI2, the counter starts counting on the internal clock and the TIF flag is set.

The delay between the rising edge on TI2 and the actual start of the counter is due to the resynchronization circuit on TI2 input.

Figure 92. Control circuit in trigger mode



Slave mode: external clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input (in reset mode, gated mode or trigger mode). It is recommended not to select ETR as TRGI through the TS bits of TIMx_SMCR register.

In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

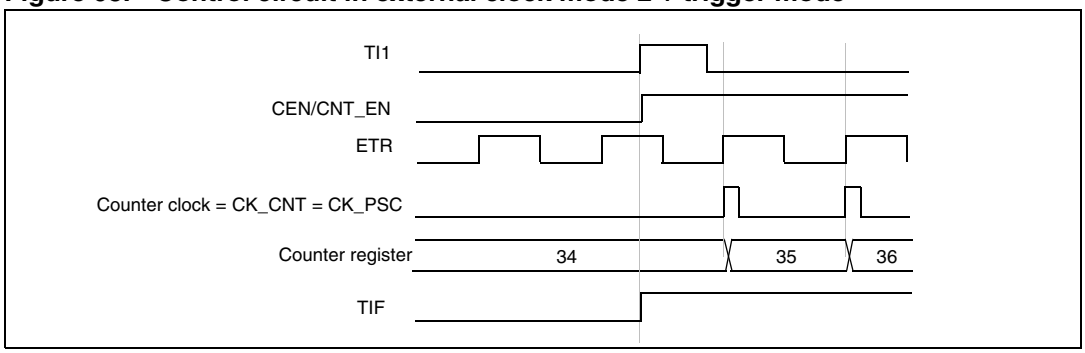
1. Configure the external trigger input circuit by programming the TIMx_SMCR register as follows:
   - ETF = 0000: no filter
   - ETPS=00: prescaler disabled
   - ETP=0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.

2. Configure the channel 1 as follows, to detect rising edges on TI:
   - IC1F=0000: no filter.
   - The capture prescaler is not used for triggering and does not need to be configured.
   - CC1S=01in TIMx_CCMR1 register to select only the input capture source
   - CC1P=0 in TIMx_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.

A rising edge on TI1 enables the counter and sets the TIF flag. The counter then counts on ETR rising edges.

The delay between the rising edge of the ETR signal and the actual reset of the counter is due to the resynchronization circuit on ETRP input.

Figure 93. Control circuit in external clock mode 2 + trigger mode



12.3.20 Timer synchronization

The TIM timers are linked together internally for timer synchronization or chaining. Refer to *Section 13.3.15: Timer synchronization on page 296* for details.

12.3.21 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module. For more details, refer to *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C*.

12.4 TIM1&TIM8 registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

12.4.1 Control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved, always read as 0

Bits 9:8 **CKD[1:0]**: Clock division.

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and the dead-time and sampling clock (t_{DTS})used by the dead-time generators and the digital filters (ETR, TIx),

00: $t_{DTS}=t_{CK_INT}$
01: $t_{DTS}=2*t_{CK_INT}$

10: $t_{DTS}=4*t_{CK_INT}$

11: Reserved, do not program this value.

Bit 7 **ARPE**: Auto-reload preload enable.

0: TIMx_ARR register is not buffered.
1: TIMx_ARR register is buffered.

Bits 6:5 **CMS[1:0]**: Center-aligned mode selection.

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

Bit 4 **DIR**: Direction.

0: Counter used as upcounter.
1: Counter used as downcounter.

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: One pulse mode.

0: Counter is not stopped at update event
1: Counter stops counting at the next update event (clearing the bit CEN).

Bit 2 **URS**: Update request source.

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled. These events can be:

– Counter overflow/underflow
– Setting the UG bit
– Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: Update disable.

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

– Counter overflow/underflow
– Setting the UG bit
– Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: Counter enable.

0: Counter disabled
1: Counter enabled

Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

12.4.2 Control register 2 (TIMx_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	TI1S	MMS[2:0]			CCDS	CCUS	Res.	CCPC
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bit 15 Reserved, always read as 0

Bit 14 **OIS4**: Output Idle state 4 (OC4 output).

refer to OIS1 bit

Bit 13 **OIS3N**: Output Idle state 3 (OC3N output).

refer to OIS1N bit

Bit 12 **OIS3**: Output Idle state 3 (OC3 output).

refer to OIS1 bit

Bit 11 **OIS2N**: Output Idle state 2 (OC2N output).

refer to OIS1N bit

Bit 10 **OIS2**: Output Idle state 2 (OC2 output).

refer to OIS1 bit

Bit 9 **OIS1N**: Output Idle state 1 (OC1N output).

0: OC1N=0 after a dead-time when MOE=0
1: OC1N=1 after a dead-time when MOE=0

Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BKR register).

Bit 8 **OIS1**: Output Idle state 1 (OC1 output).

0: OC1=0 (after a dead-time if OC1N is implemented) when MOE=0
1: OC1=1 (after a dead-time if OC1N is implemented) when MOE=0

Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BKR register).

Bit 7 **TI1S**: TI1 Selection.

0: The TIMx_CH1 pin is connected to TI1 input.
1: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Bits 6:4 **MMS[1:0]**: Master Mode Selection.

These bits allow to select the information to be sent in master mode to slave timers for synchronization (TRGO). The combination is as follows:
000: **Reset** - the UG bit from the TIMx_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.
001: **Enable** - the Counter Enable signal CNT_EN is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enable. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx_SMCR register).
010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.
011: **Compare Pulse** - The trigger output send a positive pulse when the CC1IF flag is to be set (even if it was already high), as soon as a capture or a compare match occurred. (TRGO).
100: **Compare** - OC1REF signal is used as trigger output (TRGO).
101: **Compare** - OC2REF signal is used as trigger output (TRGO).
110: **Compare** - OC3REF signal is used as trigger output (TRGO).
111: **Compare** - OC4REF signal is used as trigger output (TRGO).

Bit 3 **CCDS**: Capture/compare DMA selection.

0: CCx DMA request sent when CCx event occurs
1: CCx DMA requests sent when update event occurs

Bit 2 **CCUS**: Capture/compare control update selection.

0: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COM bit only.
1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COM bit or when an rising edge occurs on TRGI.

Note: This bit acts only on channels that have a complementary output.

Bit 1 Reserved, always read as 0

Bit 0 **CCPC**: Capture/compare preloaded control.

0: CCxE, CCxNE and OCxM bits are not preloaded
1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when COM bit is set.

Note: This bit acts only on channels that have a complementary output.

12.4.3 Slave mode control register (TIMx_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.	rw	rw	rw

Bit 15 **ETP**: External trigger polarity.

This bit selects whether ETR or $\overline{ETR}$ is used for trigger operations
0: ETR is non-inverted, active at high level or rising edge.
1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE**: External clock enable.

This bit enables External clock mode 2.
0: External clock mode 2 disabled.
1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

Note: ***1:*** *Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).*

2: *It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).*

3: *If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.*

Bits 13:12 **ETPS[1:0]**: External trigger prescaler.

External trigger signal ETRP frequency must be at most 1/4 of TIMxCLK frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF.
01: ETRP frequency divided by 2.
10: ETRP frequency divided by 4.
11: ETRP frequency divided by 8.

Bits 11:8 **ETF[3:0]**: External trigger filter.

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS}.
0001: $f_{SAMPLING}=f_{CK_INT}$, N=2.
0010: $f_{SAMPLING}=f_{CK_INT}$, N=4.
0011: $f_{SAMPLING}=f_{CK_INT}$, N=8.
0100: $f_{SAMPLING}=f_{DTS}/2$, N=6.
0101: $f_{SAMPLING}=f_{DTS}/2$, N=8.
0110: $f_{SAMPLING}=f_{DTS}/4$, N=6.
0111: $f_{SAMPLING}=f_{DTS}/4$, N=8.
1000: $f_{SAMPLING}=f_{DTS}/8$, N=6.
1001: $f_{SAMPLING}=f_{DTS}/8$, N=8.
1010: $f_{SAMPLING}=f_{DTS}/16$, N=5.
1011: $f_{SAMPLING}=f_{DTS}/16$, N=6.
1100: $f_{SAMPLING}=f_{DTS}/16$, N=8.
1101: $f_{SAMPLING}=f_{DTS}/32$, N=5.
1110: $f_{SAMPLING}=f_{DTS}/32$, N=6.
1111: $f_{SAMPLING}=f_{DTS}/32$, N=8.

Bit 7 **MSM:** Master/slave mode.

0: No action
1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS[2:0]:** Trigger selection.

This bit-field selects the trigger input to be used to synchronize the counter.
000: Internal Trigger 0 (ITR0)
001: Internal Trigger 1 (ITR1)
010: Internal Trigger 2 (ITR2)
011: Internal Trigger 3 (ITR3)
100: TI1 Edge Detector (TI1F_ED)
101: Filtered Timer Input 1 (TI1FP1)
110: Filtered Timer Input 2 (TI2FP2)
111: External Trigger input (ETRF)

See *Table 55: TIMx Internal trigger connection on page 245* for more details on ITRx meaning for each Timer.

Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.

Bit 3 Reserved, always read as 0.

Bits 2:0 **SMS** Slave mode selection.

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description.
000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.
001: Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.
010: Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.
011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.
100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.
101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.
110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.
111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS='100'). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

Table 55. TIMx Internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM1	TIM5	TIM2	TIM3	TIM4
TIM8	TIM1	TIM2	TIM4	TIM5

12.4.4 DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, always read as 0.

Bit 14 **TDE**: *Trigger DMA request enable.*

0: Trigger DMA request disabled.
1: Trigger DMA request enabled.

Bit 13 **COMDE**: *COM DMA request enable.*

0: COM DMA request disabled.
1: COM DMA request enabled.

Bit 12 **CC4DE**: *Capture/Compare 4 DMA request enable.*

0: CC4 DMA request disabled.
1: CC4 DMA request enabled.

Bit 11 **CC3DE**: *Capture/Compare 3 DMA request enable.*

0: CC3 DMA request disabled.
1: CC3 DMA request enabled.

Bit 10 **CC2DE**: *Capture/Compare 2 DMA request enable.*

0: CC2 DMA request disabled.
1: CC2 DMA request enabled.

Bit 9 **CC1DE**: *Capture/Compare 1 DMA request enable.*

0: CC1 DMA request disabled.
1: CC1 DMA request enabled.

Bit 8 **UDE**: *Update DMA request enable.*

0: Update DMA request disabled.
1: Update DMA request enabled.

Bit 7 **BIE**: *Break interrupt enable.*

0: Break interrupt disabled.
1: Break interrupt enabled.

Bit 6 **TIE**: *Trigger interrupt enable.*

0: Trigger interrupt disabled.
1: Trigger interrupt enabled.

Bit 5 **COMIE**: *COM interrupt enable.*

0: COM interrupt disabled.
1: COM interrupt enabled.

Bit 4 **CC4IE**: *Capture/Compare 4 interrupt enable.*

0: CC4 interrupt disabled.
1: CC4 interrupt enabled.

Bit 3 **CC3IE**: *Capture/Compare 3 interrupt enable.*

0: CC3 interrupt disabled.
1: CC3 interrupt enabled.

Bit 2 **CC2IE**: *Capture/Compare 2 interrupt enable.*

0: CC2 interrupt disabled.
1: CC2 interrupt enabled.

Bit 1 **CC1IE**: *Capture/Compare 1 interrupt enable.*

0: CC1 interrupt disabled.
1: CC1 interrupt enabled.

Bit 0 **UIE**: *Update interrupt enable.*

0: Update interrupt disabled.
1: Update interrupt enabled.

12.4.5 Status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
Res.			rc_w0	rc_w0	rc_w0	rc_w0	Res.	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 15:13 Reserved, always read as 0.

Bit 12 **CC4OF**: *Capture/Compare 4 Overcapture Flag.*

refer to CC1OF description

Bit 11 **CC3OF**: *Capture/Compare 3 Overcapture Flag.*

refer to CC1OF description

Bit 10 **CC2OF**: *Capture/Compare 2 Overcapture Flag.*

refer to CC1OF description

Bit 9 **CC1OF**: *Capture/Compare 1 Overcapture Flag.*

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to ‘0’.
0: No overcapture has been detected.
1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bit 8 Reserved, always read as 0.

Bit 7 **BIF**: *Break interrupt Flag.*

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.
0: No break event occurred.
1: An active level has been detected on the break input.

Bit 6 **TIF**: *Trigger interrupt Flag.*

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode, both edges in case gated mode is selected). It is cleared by software.
0: No trigger event occurred.
1: Trigger interrupt pending.

Bit 5 **COMIF**: *COM interrupt Flag.*

This flag is set by hardware on COM event (when Capture/compare Control bits - CCxE, CCxNE, OCxM - have been updated). It is cleared by software.
0: No COM event occurred.
1: COM interrupt pending.

Bit 4 **CC4IF**: *Capture/Compare 4 interrupt Flag.*

refer to CC1IF description

Bit 3 **CC3IF**: *Capture/Compare 3 interrupt Flag.*

refer to CC1IF description

Bit 2 **CC2IF**: *Capture/Compare 2 interrupt Flag.*

refer to CC1IF description

Bit 1 **CC1IF**: *Capture/Compare 1 interrupt Flag.*

If channel CC1 is configured as output:
This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description). It is cleared by software.
0: No match.
1: The content of the counter TIMx_CNT has matched the content of the TIMx_CCR1 register.

If channel CC1 is configured as input:
This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.
0: No input capture occurred.
1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity).

Bit 0 **UIF**: *Update interrupt Flag.*

This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value (update if REP_CNT=0) and if the UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by a trigger event (refer to *Section 12.4.3: Slave mode control register (TIMx_SMCR)*), if URS=0 and UDIS=0 in the TIMx_CR1 register.

12.4.6 Event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG
Res.								w	w	w	w	w	w	w	w

Bits 15:8 Reserved, always read as 0.

Bit 7 **BG**: *Break Generation.*

This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action.
1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

Bit 6 **TG**: *Trigger Generation.*

This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action.
1: The TIF flag is set in TIMx_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 **COMG**: *Capture/Compare Control Update Generation.*

This bit can be set by software, it is automatically cleared by hardware
0: No action
1: When CCPC bit is set, it allows to update CCxE, CCxNE and OCxM bits

Note: This bit acts only on channels having a complementary output.

Bit 4 **CC4G**: *Capture/Compare 4 Generation.*

refer to CC1G description

Bit 3 **CC3G**: *Capture/Compare 3 Generation.*

refer to CC1G description

Bit 2 **CC2G**: *Capture/Compare 2 Generation.*

refer to CC1G description

Bit 1 **CC1G**: *Capture/Compare 1 Generation.*

This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action.
1: A capture/compare event is generated on channel 1:
If channel CC1 is configured as output:
CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.
If channel CC1 is configured as input:
The current value of the counter is captured in TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: *Update Generation.*

This bit can be set by software, it is automatically cleared by hardware.
0: No action.
1: Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

12.4.7 Capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]		OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Output compare mode:

Bit 15 **OC2CE:** *Output Compare 2 Clear Enable*

Bits 14:12 **OC2M[2:0]**: *Output Compare 2 Mode.*

Bit 11 **OC2PE**: *Output Compare 2 Preload enable.*

Bit 10 **OC2FE**: *Output Compare 2 Fast enable.*

Bits 9:8 **CC2S[1:0]**: *Capture/Compare 2 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC2 channel is configured as output.
01: CC2 channel is configured as input, IC2 is mapped on TI2.
10: CC2 channel is configured as input, IC2 is mapped on TI1.
11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = ‘0’ in TIMx_CCER).

Bit 7 **OC1CE:** *Output Compare 1Clear Enable*

OC1CE: Output Compare 1 Clear Enable
0: OC1Ref is not affected by the ETRF Input
1: OC1Ref is cleared as soon as a High level is detected on ETRF input

Bits 6:4 **OC1M**: *Output Compare 1 Mode.*

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.
000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.
001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.
100: Force inactive level - OC1REF is forced low.
101: Force active level - OC1REF is forced high.
110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF=‘0’) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=’1’).
111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

*Note: **1:** These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S=’00’ (the channel is configured in output).*

__2:__ In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.

Bit 3 **OC1PE**: *Output Compare 1 Preload enable.*

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.
1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

*Note: **1:** These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S=’00’ (the channel is configured in output).*

__2:__ The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.

Bit 2 **OC1FE**: *Output Compare 1 Fast enable.*

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.
0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.
1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: *Capture/Compare 1 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC1 channel is configured as output.
01: CC1 channel is configured as input, IC1 is mapped on TI1.
10: CC1 channel is configured as input, IC1 is mapped on TI2.
11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = ‘0’ in TIMx_CCER).

Input capture mode

Bits 15:12 **IC2F**: *Input Capture 2 Filter.*

Bits 11:10 **IC2PSC[1:0]**: *Input Capture 2 Prescaler.*

Bits 9:8 **CC2S**: *Capture/Compare 2 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC2 channel is configured as output.
01: CC2 channel is configured as input, IC2 is mapped on TI2.
10: CC2 channel is configured as input, IC2 is mapped on TI1.
11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCER).

Bits 7:4 **IC1F[3:0]**: *Input Capture 1 Filter.*

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS}.
0001: $f_{SAMPLING}=f_{CK_INT}$, N=2.
0010: $f_{SAMPLING}=f_{CK_INT}$, N=4.
0011: $f_{SAMPLING}=f_{CK_INT}$, N=8.
0100: $f_{SAMPLING}=f_{DTS}/2$, N=6.
0101: $f_{SAMPLING}=f_{DTS}/2$, N=8.
0110: $f_{SAMPLING}=f_{DTS}/4$, N=6.
0111: $f_{SAMPLING}=f_{DTS}/4$, N=8.
1000: $f_{SAMPLING}=f_{DTS}/8$, N=6.
1001: $f_{SAMPLING}=f_{DTS}/8$, N=8.
1010: $f_{SAMPLING}=f_{DTS}/16$, N=5.
1011: $f_{SAMPLING}=f_{DTS}/16$, N=6.
1100: $f_{SAMPLING}=f_{DTS}/16$, N=8.
1101: $f_{SAMPLING}=f_{DTS}/32$, N=5.
1110: $f_{SAMPLING}=f_{DTS}/32$, N=6.
1111: $f_{SAMPLING}=f_{DTS}/32$, N=8.

Bits 3:2 **IC1PSC**: *Input Capture 1 Prescaler.*

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).
The prescaler is reset as soon as CC1E='0' (TIMx_CCER register).
00: no prescaler, capture is done each time an edge is detected on the capture input.
01: capture is done once every 2 events.
10: capture is done once every 4 events.
11: capture is done once every 8 events.

Bits 1:0 **CC1S**: *Capture/Compare 1 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC1 channel is configured as output.
01: CC1 channel is configured as input, IC1 is mapped on TI1.
10: CC1 channel is configured as input, IC1 is mapped on TI2.
11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).

12.4.8 Capture/compare mode register 2 (TIMx_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4 CE	OC4M[2:0]			OC4 PE	OC4 FE	CC4S[1:0]		OC3 CE.	OC3M[2:0]			OC3 PE	OC3 FE	CC3S[1:0]	
IC4F[3:0]				IC4PSC[1:0]				IC3F[3:0]				IC3PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Output Compare mode

Bit 15 **OC4CE:** Output Compare 4 Clear Enable

Bits 14:12 **OC4M**: *Output Compare 4 Mode.*

Bit 11 **OC4PE**: *Output Compare 4 Preload enable.*

Bit 10 **OC4FE**: *Output Compare 4 Fast enable.*

Bits 9:8 **CC4S**: *Capture/Compare 4 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC4 channel is configured as output.
01: CC4 channel is configured as input, IC4 is mapped on TI4.
10: CC4 channel is configured as input, IC4 is mapped on TI3.
11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx_CCER).

Bit 7 **OC3CE:** Output Compare 3 Clear Enable

Bits 6:4 **OC3M**: *Output Compare 3 Mode.*

Bit 3 **OC3PE**: *Output Compare 3 Preload enable.*

Bit 2 **OC3FE**: *Output Compare 3 Fast enable.*

Bits 1:0 **CC3S**: *Capture/Compare 3 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC3 channel is configured as output.
01: CC3 channel is configured as input, IC3 is mapped on TI3.
10: CC3 channel is configured as input, IC3 is mapped on TI4.
11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx_CCER).

Input capture mode

Bits 15:12 **IC4F**: *Input Capture 4 Filter.*

Bits 11:10 **IC4PSC**: *Input Capture 4 Prescaler.*

Bits 9:8 **CC4S**: *Capture/Compare 4 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC4 channel is configured as output.
01: CC4 channel is configured as input, IC4 is mapped on TI4.
10: CC4 channel is configured as input, IC4 is mapped on TI3.
11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx_CCER).

Bits 7:4 **IC3F**: *Input Capture 3 Filter.*

Bits 3:2 **IC3PSC**: *Input Capture 3 Prescaler.*

Bits 1:0 **CC3S**: *Capture/Compare 3 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC3 channel is configured as output.
01: CC3 channel is configured as input, IC3 is mapped on TI3.
10: CC3 channel is configured as input, IC3 is mapped on TI4.
11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx_CCER).

12.4.9 Capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
Res.		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:14 Reserved, always read as 0.

Bit 13 **CC4P**: *Capture/Compare 4 output Polarity.*

refer to CC1P description

Bit 12 **CC4E**: *Capture/Compare 4 output enable.*

refer to CC1E description

Bit 11 **CC3NP**: *Capture/Compare 3 Complementary output Polarity.*

refer to CC1NP description

Bit 10 **CC3NE**: *Capture/Compare 3 Complementary output enable.*

refer to CC1NE description

Bit 9 **CC3P**: *Capture/Compare 3 output Polarity.*

refer to CC1P description

Bit 8 **CC3E**: *Capture/Compare 3 output enable.*

refer to CC1E description

Bit 7 **CC2NP**: *Capture/Compare 2 Complementary output Polarity.*

refer to CC1NP description

Bit 6 **CC2NE**: *Capture/Compare 2 Complementary output enable.*

refer to CC1NE description

Bit 5 **CC2P**: *Capture/Compare 2 output Polarity.*

refer to CC1P description

Bit 4 **CC2E**: *Capture/Compare 2 output enable.*

refer to CC1E description

Bit 3 **CC1NP**: *Capture/Compare 1 Complementary output Polarity.*

0: OC1N active high.
1: OC1N active low.

Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S="00" (the channel is configured in output).

Bit 2 **CC1NE**: *Capture/Compare 1 Complementary output enable.*

0: Off - OC1N is not active. OC1N level is then function of MOE, OSSI, OSSR, OIS1, OIS1N and CC1E bits.
1: On - OC1N signal is output on the corresponding output pin depending on MOE, OSSI, OSSR, OIS1, OIS1N and CC1E bits.

Bit 1 **CC1P**: *Capture/Compare 1 output Polarity.*

CC1 channel configured as output:
0: OC1 active high.
1: OC1 active low.
CC1 channel configured as input:
This bit selects whether IC1 or IC1 is used for trigger or capture operations.
0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted.
1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.

Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*

CC1 channel configured as output:
0: Off - OC1 is not active. OC1 level is then function of MOE, OSSI, OSSR, OIS1, OIS1N and CC1NE bits.
1: On - OC1 signal is output on the corresponding output pin depending on MOE, OSSI, OSSR, OIS1, OIS1N and CC1NE bits.

CC1 channel configured as input:
This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.
0: Capture disabled.
1: Capture enabled.

Table 56. Output control bits for complementary OCx and OCxN channels with break feature

Control bits					Output states[1]	
MOE bit	**OSSI bit**	**OSSR bit**	**CCxE bit**	**CCxNE bit**	**OCx output state**	**OCxN output state**
1	X	0	0	0	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
		0	0	1	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
		0	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP, OCx_EN=1	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Off-State (output enabled with inactive state) OCxN=CCxNP, OCxN_EN=1
		1	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
0	0	X	0	0	Output Disabled (not driven by the timer) Asynchronously: OCx=CCxP, OCx_EN=0, OCxN=CCxNP, OCxN_EN=0 Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state.	
	0		0	1		
	0		1	0		
	0		1	1		
	1		0	0		
	1		0	1	Off-State (output enabled with inactive state) Asynchronously: OCx=CCxP, OCx_EN=1, OCxN=CCxNP, OCxN_EN=1 Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state	
	1		1	0		
	1		1	1		

1. When both outputs of a channel are not used (CCxE = CCxNE = 0), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

Note: The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO and AFIO registers.

12.4.10 Counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CNT[15:0]**: *Counter Value.*

12.4.11 Prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: *Prescaler Value.*

The counter clock frequency (CK_CNT) is equal to f_{CK_PSC} / (PSC[15:0] + 1).

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

12.4.12 Auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: *Prescaler Value.*

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 12.3.1: Time-base unit on page 202* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

12.4.13 Repetition counter register (TIMx_RCR)

Address offset: 0x30

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								REP[7:0]							
Res.								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved, always read as 0.

Bits 7:0 **REP[7:0]**: *Repetition Counter Value.*

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP_CNT is reloaded with REP value only at the repetition update event U_RC, any write to the TIMx_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to:
- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode.

12.4.14 Capture/compare register 1 (TIMx_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: *Capture/Compare 1 Value).*

If channel CC1 is configured as output:
CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC1 output.

If channel CC1 is configured as input:
CCR1 is the counter value transferred by the last input capture 1 event (IC1).

12.4.15 Capture/compare register 2 (TIMx_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: *Capture/Compare 2 Value.*

If channel CC2 is configured as output:

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC2 output.

If channel CC2 is configured as input:

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

12.4.16 Capture/compare register 3 (TIMx_CCR3)

Address offset: 0x3C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR3[15:0]**: *Capture/Compare Value.*

If channel CC3 is configured as output:

CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR3 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC3 output.

If channel CC3 is configured as input:

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

12.4.17 Capture/compare register 4 (TIMx_CCR4)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]**: *Capture/Compare Value.*

If channel CC4 is configured as output:

CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR4 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC4 output.

If channel CC4 is configured as input:

CCR4 is the counter value transferred by the last input capture 4 event (IC4).

12.4.18 Break and dead-time register (TIMx_BDTR)

Address offset: 0x44

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: As the bits AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.

Bit 15 **MOE**: *Main Output enable.*

This bit is cleared asynchronously by hardware as soon as the break input is active. It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.
0: OC and OCN outputs are disabled or forced to idle state.
1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx_CCER register).
See OC/OCN enable description for more details (*Section 12.4.9: Capture/compare enable register (TIMx_CCER) on page 254*).

Bit 14 **AOE**: *Automatic Output enable.*

0: MOE can be set only by software
1: MOE can be set by software or automatically at the next update event (if the break input is not be active)

Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 13 **BKP**: *Break Polarity.*

0: Break input BRK is active low
1: Break input BRK is active high

Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 12 **BKE**: *Break enable.*

0: Break inputs (BRK and BRK_ACTH) disabled
1; Break inputs (BRK and BRK_ACTH) enabled

Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 11 **OSSR**: *Off-State Selection for Run mode.*

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.
See OC/OCN enable description for more details (*Section 12.4.9: Capture/compare enable register (TIMx_CCER) on page 254*).
0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0).
1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1. Then, OC/OCN enable output signal=1

Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 10 **OSSI**: *Off-State Selection for Idle mode.*

This bit is used when MOE=0 on channels configured as outputs.
See OC/OCN enable description for more details (*Section 12.4.9: Capture/compare enable register (TIMx_CCER) on page 254*).
0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0).
1: When inactive, OC/OCN outputs are forced first with their idle level as soon as CCxE=1 or CCxNE=1. OC/OCN enable output signal=1)

Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx_BDTR register).

Bits 9:8 **LOCK[1:0]**: *Lock Configuration.*

These bits offer a write protection against software errors.
00: LOCK OFF - No bit is write protected.
01: LOCK Level 1 = DTG bits in TIMx_BDTR register, OISx and OISxN bits in TIMx_CR2 register and BKE/BKP/AOE bits in TIMx_BDTR register can no longer be written.
10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.
11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

Note: The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.

Bits 7:0 **DTG[7:0]**: *Dead-Time Generator set-up.*

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.
DTG[7:5]=0xx => DT=DTG[7:0]x t_{dtg} with $t_{dtg}=t_{DTS}$.
DTG[7:5]=10x => DT=(64+DTG[5:0])xt_{dtg} with $T_{dtg}=2xt_{DTS}$.
DTG[7:5]=110 => DT=(32+DTG[4:0])xt_{dtg} with $T_{dtg}=8xt_{DTS}$.
DTG[7:5]=111 => DT=(32+DTG[4:0])xt_{dtg} with $T_{dtg}=16xt_{DTS}$.
Example if T_{DTS}=125ns (8MHz), dead-time possible values are:
0 to 15875 ns by 125 ns steps,
16 us to 31750 ns by 250 ns steps,
32 us to 63us by 1 us steps,
64 us to 126 us by 2 us steps

Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

12.4.19 DMA control register (TIMx_DCR)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			DBL[4:0]					Reserved			DBA[4:0]				
Res.			rw	rw	rw	rw	rw	Res.			rw	rw	rw	rw	rw

Bits 15:13 Reserved, always read as 0

Bits 12:8 **DBL[4:0]**: *DMA Burst Length.*

This 5-bit vector defines the length of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address), i.e. the number of transfers. Transfers can be in half-words or in bytes (see example below).
00000: 1 transfer,
00001: 2 transfers,
00010: 3 transfers,
...
10001: 18 transfers.

Example: Let us consider the following transfer: DBL = 7 bytes & DBA = TIM2_CR1.

– If DBL = 7 bytes and DBA = TIM2_CR1 represents the address of the byte to be transferred, the address of the transfer should be given by the following equation:

(TIMx_CR1 address) + DBA + (DMA index), where DMA index = DBL

In this example, 7 bytes are added to (TIMx_CR1 address) + DBA, which gives us the address from/to which the data will be copied. In this case, the transfer is done to 7 registres starting from the following address: (TIMx_CR1 address) + DBA

According to the configuration of the DMA Data Size, several cases may occur:

– If you configure the DMA Data Size in half-words, 16-bit data will be transferred to each of the 7 registers.

– If you configure the DMA Data Size in bytes, the data will aslo be transferred to 7 registers: the first register will contain the first MSB byte, the second register, the first LSB byte and so on. So with the transfer Timer, you also have to specify the size of data transferred by DMA.

Bits 7:5 Reserved, always read as 0

Bits 4:0 **DBA[4:0]**: *DMA Base Address.*

This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register.
Example:
00000: TIMx_CR1,
00001: TIMx_CR2,
00010: TIMx_SMCR,
...

12.4.20 DMA address for full transfer (TIMx_DMAR)

Address offset: 0x4C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAB[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **DMAB[15:0]**: *DMA register for burst accesses.*

A read or write access to the DMAR register accesses the register located at the address:

"(TIMx_CR1 address) + DBA + (DMA index)" in which:

TIMx_CR1 address is the address of the control register 1,

DBA is the DMA base address configured in TIMx_DCR register,

DMA index is the offset automatically controlled by the DMA transfer, depending on the length of the transfer DBL in the TIMx_DCR register.

12.4.21 TIM1&TIM8 register map

TIM1&TIM8 registers are mapped as 16-bit addressable registers as described in the table below:

Table 57. TIM1&TIM8 Register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**TIMx_CR1**	Reserved																						CKD [1:0]		ARPE	CMS [1:0]		DIR	OPM	URS	UDIS	CEN
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x04	**TIMx_CR2**	Reserved																	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	TI1S	MMS[2:0]			CCDS	CCUS	Reserved	CCPC
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0		0
0x08	**TIMx_SMCR**	Reserved																ETP	ECE	ETPS [1:0]		ETF[3:0]				MSM	TS[2:0]			Reserved	SMS[2:0]		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
0x0C	**TIMx_DIER**	Reserved																	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**TIMx_SR**	Reserved																			CC4OF	CC3OF	CC2OF	CC1OF	Reserved	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
	Reset value																				0	0	0	0		0	0	0	0	0	0	0	0
0x14	**TIMx_EGR**	Reserved																								BG	TG	COM	CC4G	CC3G	CC2G	CC1G	UG
	Reset value																									0	0	0	0	0	0	0	0
0x18	**TIMx_CCMR1** *Output Compare mode*	Reserved																OC2CE	OC2M [2:0]			OC2PE	OC2FE	CC2S [1:0]		OC1CE	OC1M [2:0]			OC1PE	OC1FE	CC1S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TIMx_CCMR1 *Input Capture mode*	Reserved																IC2F[3:0]				IC2 PSC [1:0]		CC2S [1:0]		IC1F[3:0]				IC1 PSC [1:0]		CC1S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 57. TIM1&TIM8 Register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	**TIMx_CCMR2** *Output Compare mode*	Reserved																O24CE	OC4M [2:0]			OC4PE	OC4FE	CC4S [1:0]		OC3CE	OC3M [2:0]			OC3PE	OC3FE	CC3S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TIMx_CCMR2 *Input Capture mode*	Reserved																IC4F[3:0]				IC4 PSC [1:0]		CC4S [1:0]		IC3F[3:0]				IC3 PSC [1:0]		CC3S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	**TIMx_CCER**	Reserved																		CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	**TIMx_CNT**	Reserved																CNT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	**TIMx_PSC**	Reserved																PSC[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**TIMx_ARR**	Reserved																ARR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x30	**TIMx_RCR**	Reserved																								REP[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x34	**TIMx_CCR1**	Reserved																CCR1[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x38	**TIMx_CCR2**	Reserved																CCR2[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	**TIMx_CCR3**	Reserved																CCR3[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x40	**TIMx_CCR4**	Reserved																CCR4[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44	**TIMx_BDTR**	Reserved																MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK [1:0]		DT[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x48	**TIMx_DCR**	Reserved																			DBL[4:0]					Reserved				DBA[4:0]			
	Reset value																				0	0	0	0	0					0	0	0	0
0x4C	**TIMx_DMAR**	Reserved																DMAB[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

13 General-purpose timer (TIMx)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

13.1 TIMx introduction

The general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare and PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

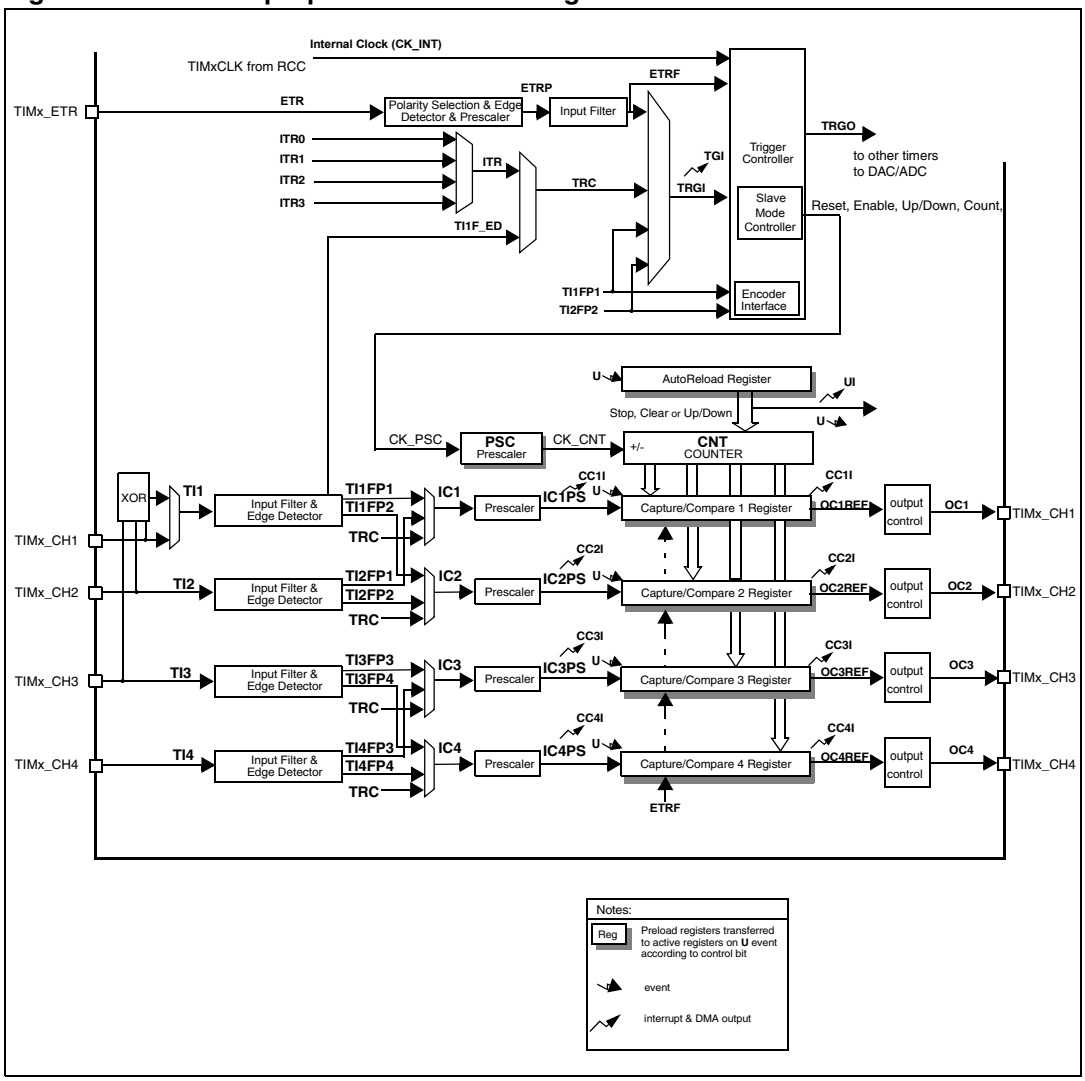
The timers are completely independent, and do not share any resources. They can be synchronized together as described in *Section 13.3.15*.

13.2 TIMx main features

General purpose TIMx (TIM2, TIM3, TIM4 and TIM5) timer features include:

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65535.
- Up to 4 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge and Center-aligned mode)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers between them.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
- Supports incremental (quadrature) encoder and hall-sensor cicuitry for positioning purposes
- Trigger intput for external clock or cycle-by-cycle current management

Figure 94. General-purpose timer block diagram



13.3 TIMx functional description

13.3.1 Time-base unit

The main block of the programmable timer is a 16-bit counter with its related auto-reload register. The counter can count up, down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC):
- Auto-Reload Register (TIMx_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Note that the actual counter enable signal CNT_EN is set 1 clock cycle after CEN.

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It 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.

Figure 95 and *Figure 96* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

Figure 95. Counter timing diagram with prescaler division change from 1 to 2

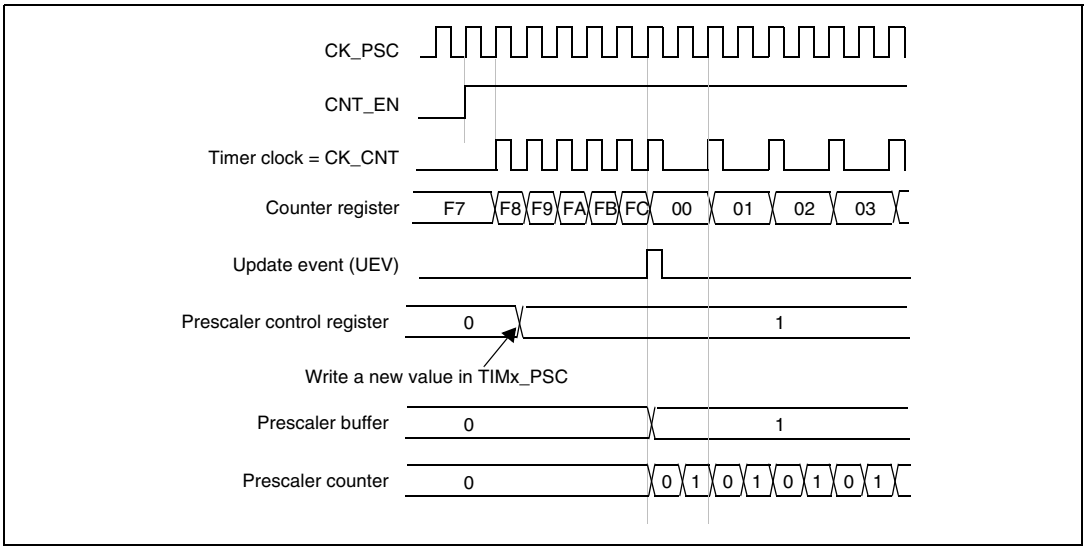
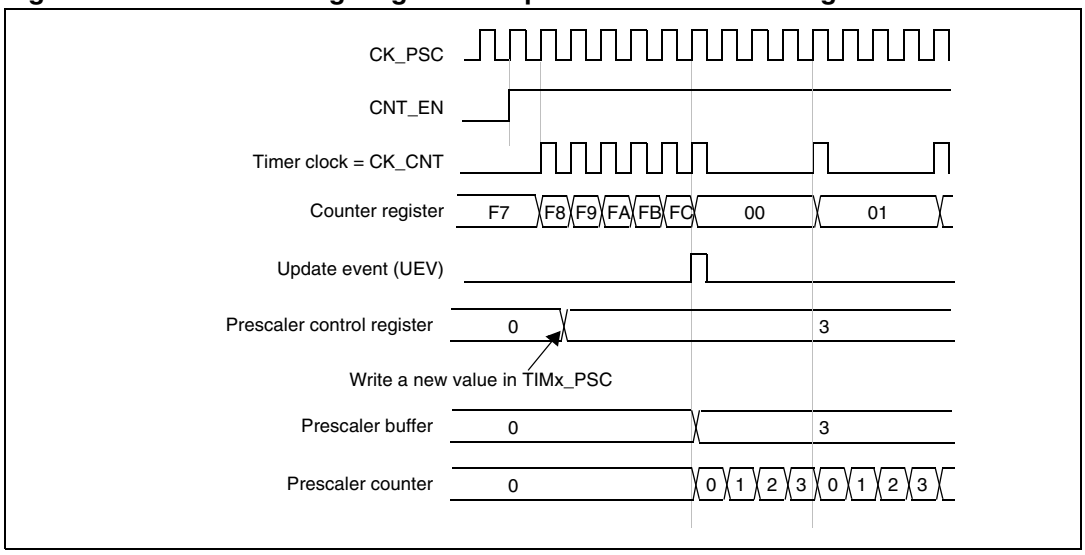


Figure 96. Counter timing diagram with prescaler division change from 1 to 4



13.3.2 Counter modes

upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

An Update event can be generated at each counter overflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller).

The UEV event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 97. Counter timing diagram, internal clock divided by 1

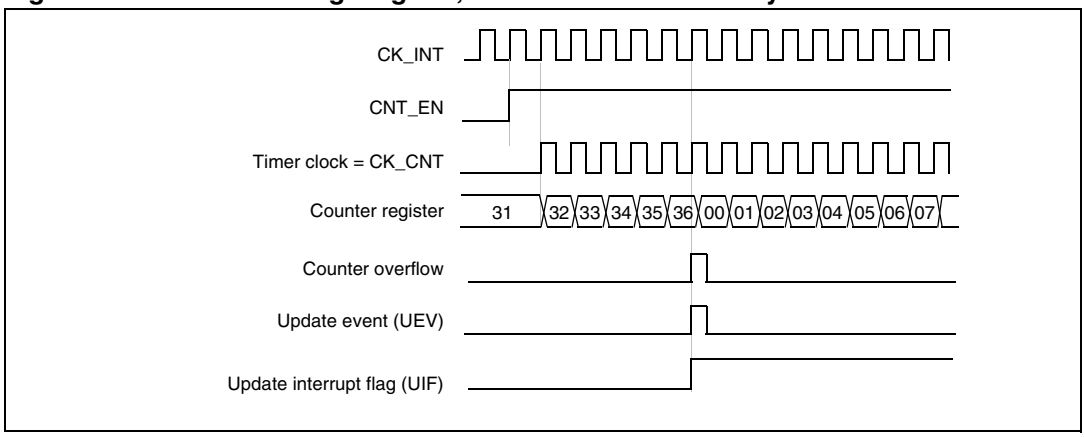


Figure 98. Counter timing diagram, internal clock divided by 2

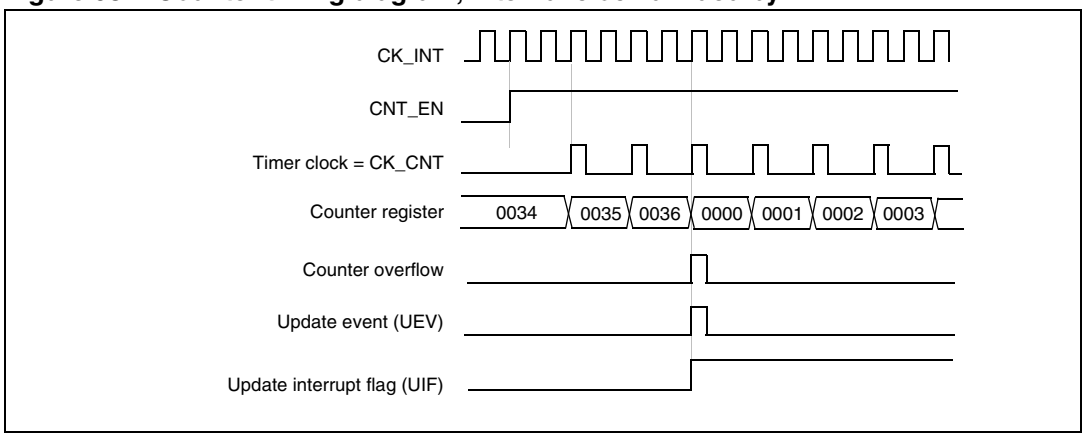


Figure 99. Counter timing diagram, internal clock divided by 4

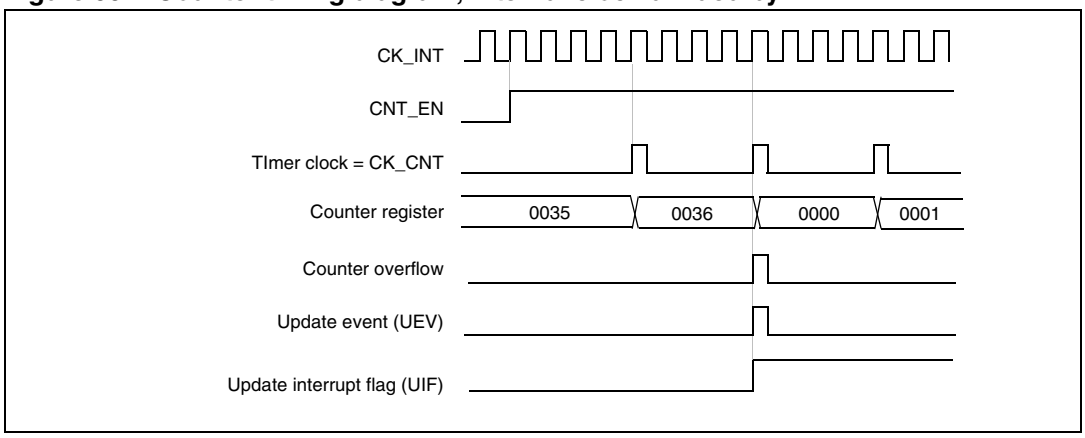


Figure 100. Counter timing diagram, internal clock divided by N

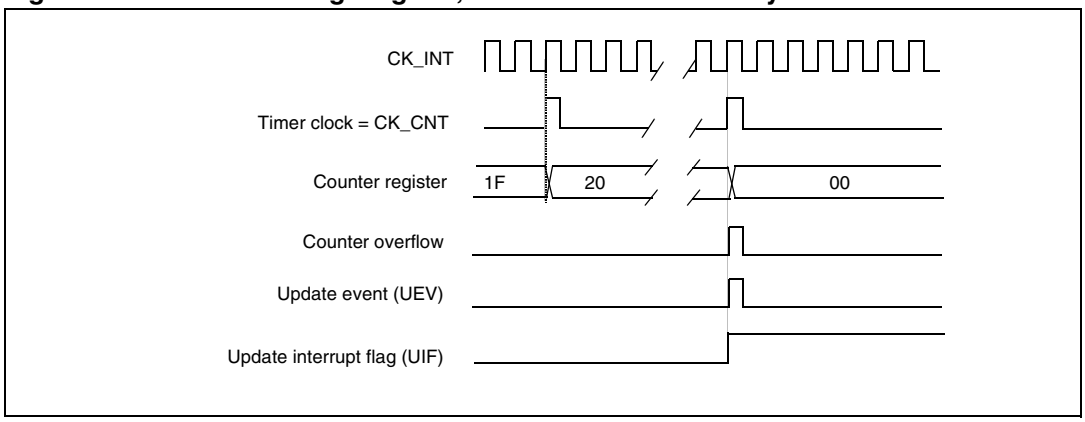


Figure 101. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded)

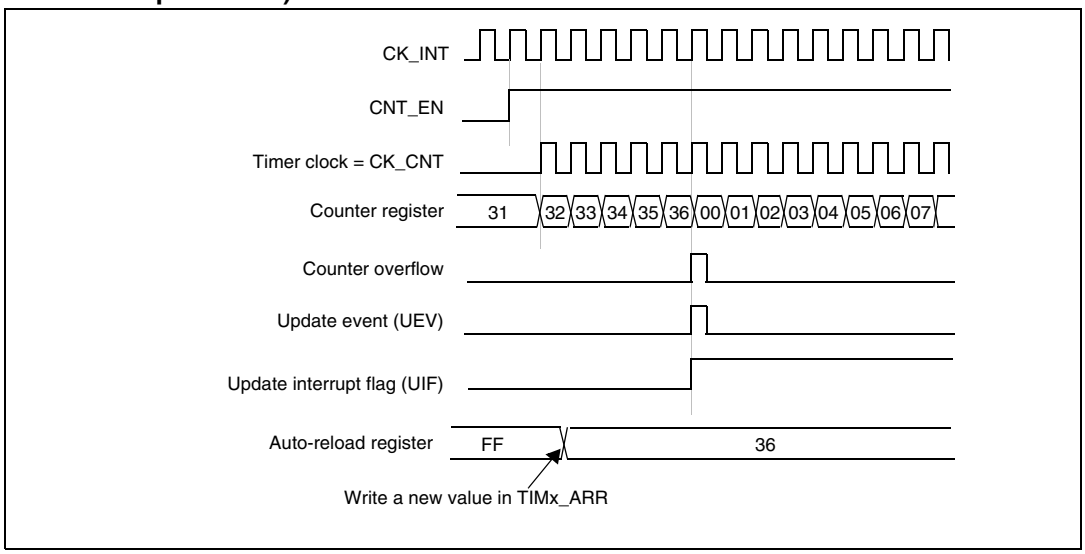
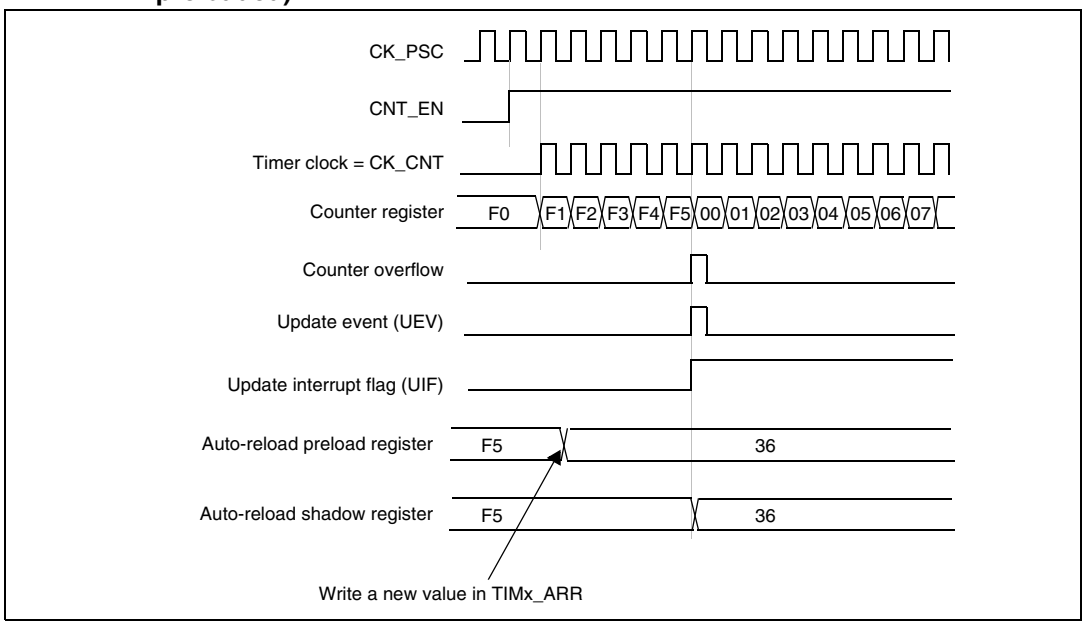


Figure 102. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded)



Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

An Update event can be generate at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller)

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 103. Counter timing diagram, internal clock divided by 1

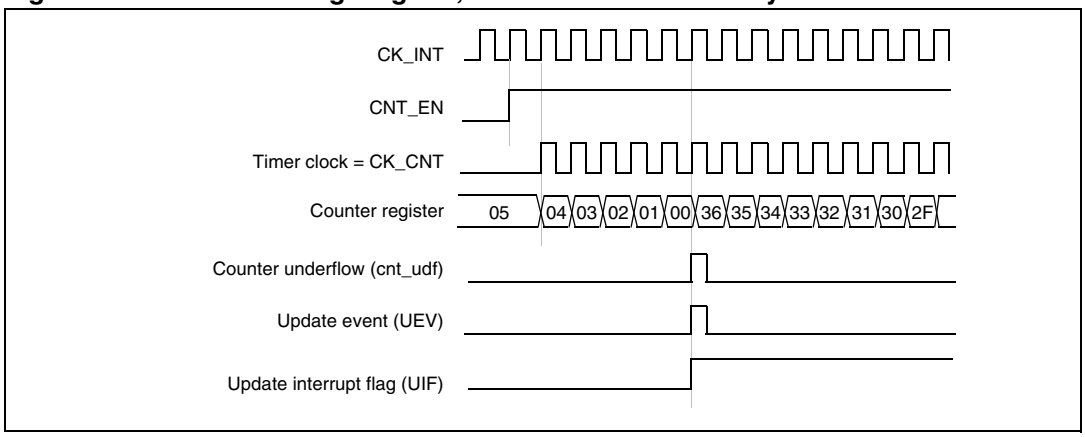


Figure 104. Counter timing diagram, internal clock divided by 2

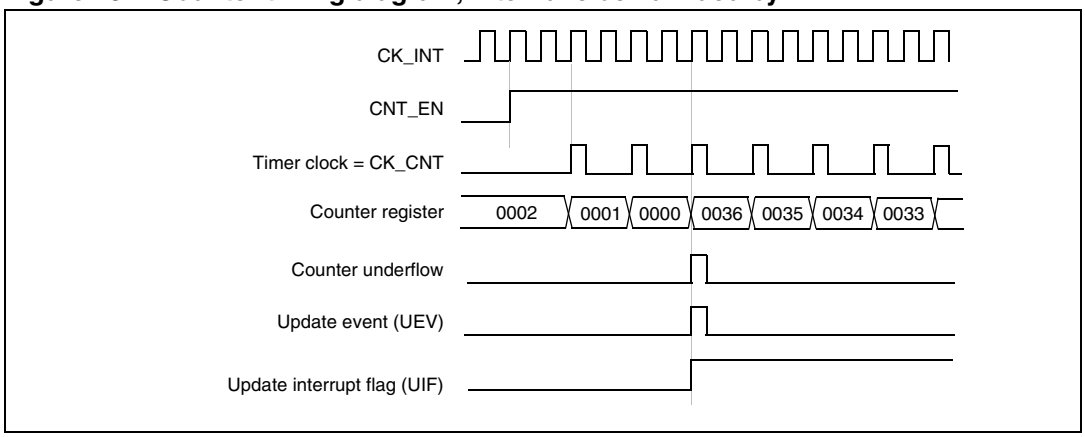


Figure 105. Counter timing diagram, internal clock divided by 4

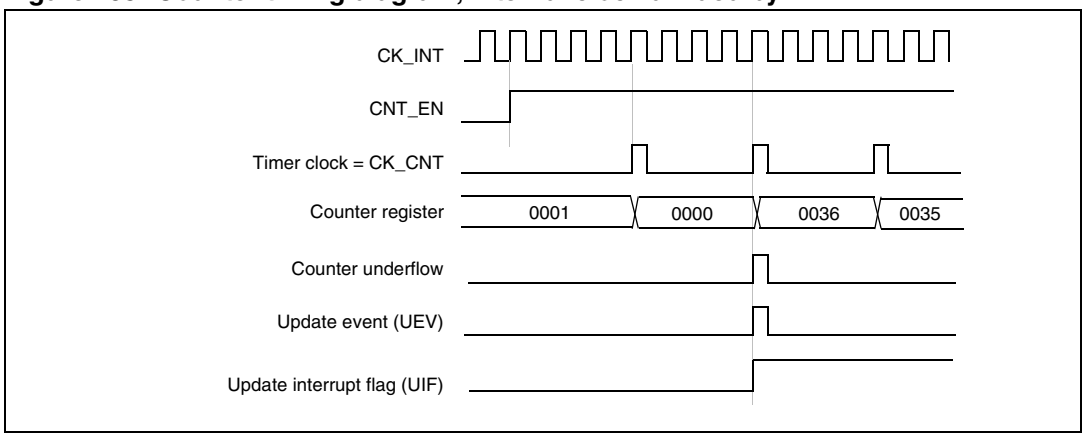


Figure 106. Counter timing diagram, internal clock divided by N

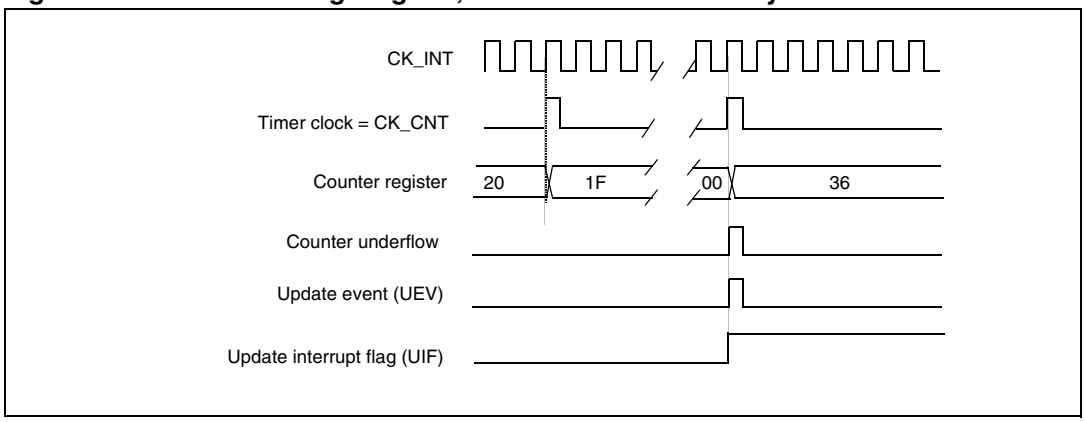
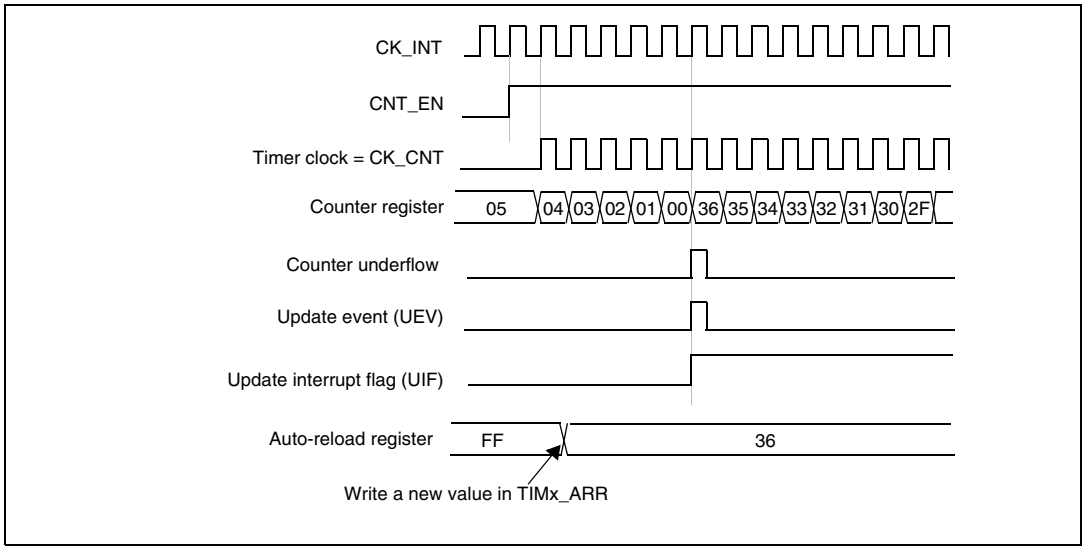


Figure 107. Counter timing diagram, Update event when repetition counter is not used



Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

In this mode, the direction bit (DIR from TIMx_CR1 register) cannot be written. It is updated by hardware and gives the current direction of the counter.

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

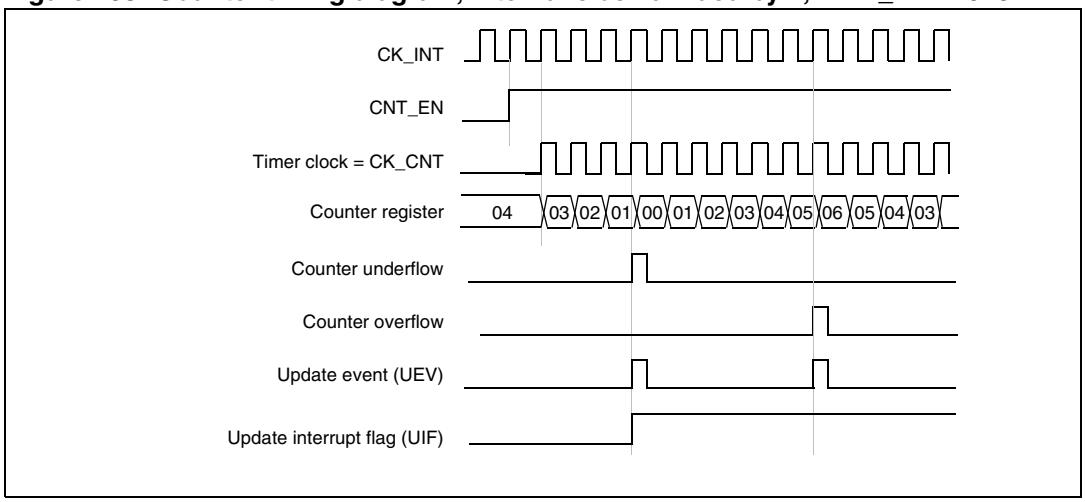
In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupt when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

Figure 108. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6



1. Here, center-aligned mode 1 is used (for more details refer to *Section 13.4.1: Control register 1 (TIMx_CR1) on page 302*).

Figure 109. Counter timing diagram, internal clock divided by 2

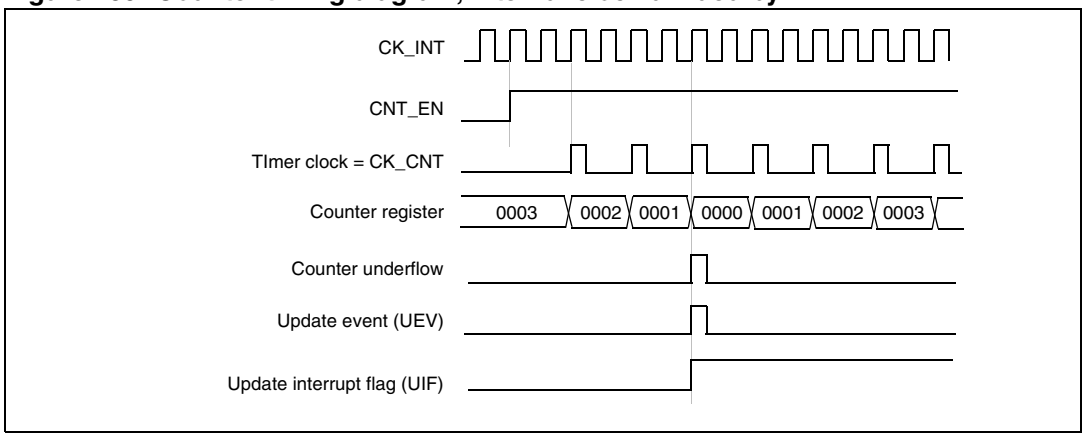
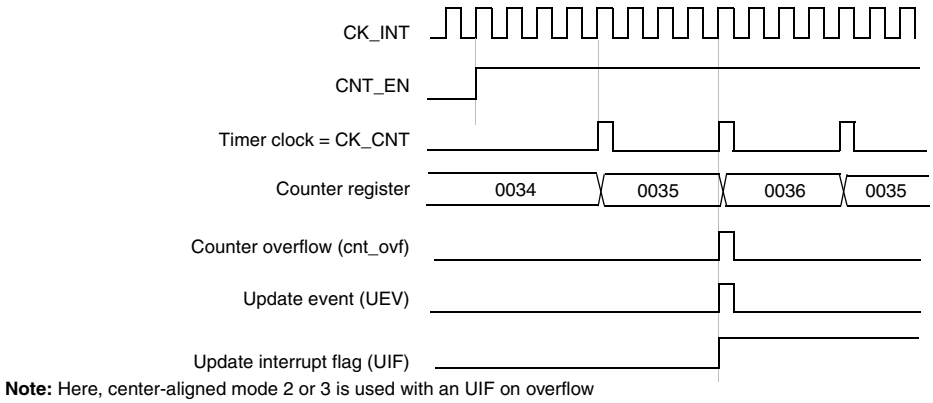


Figure 110. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36



Note: Here, center-aligned mode 2 or 3 is used with an UIF on overflow

Figure 111. Counter timing diagram, internal clock divided by N

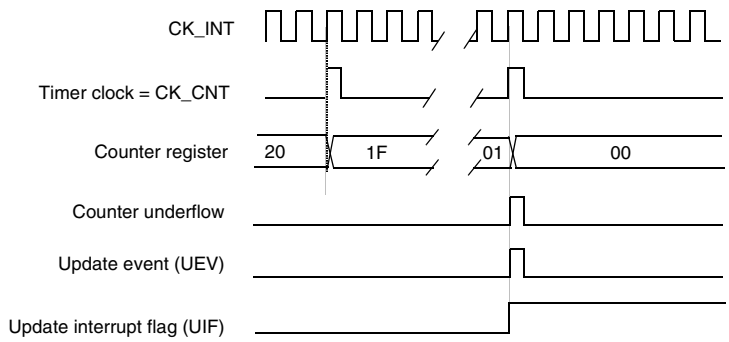


Figure 112. Counter timing diagram, Update event with ARPE=1 (counter underflow)

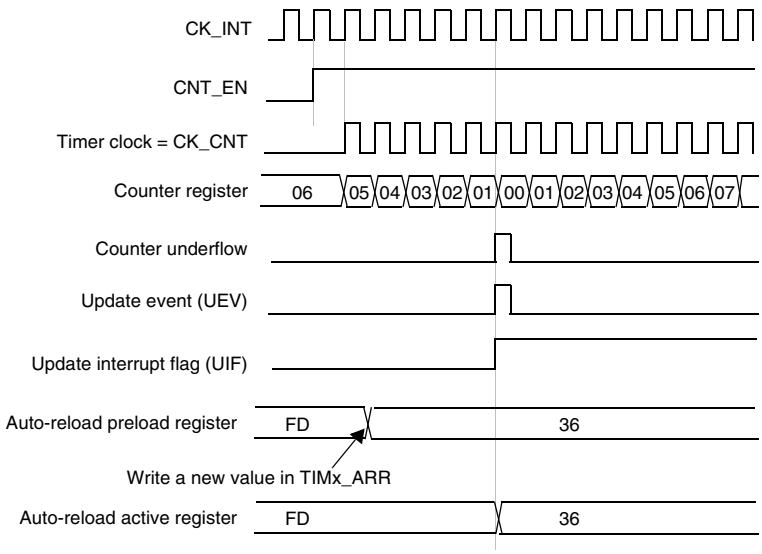
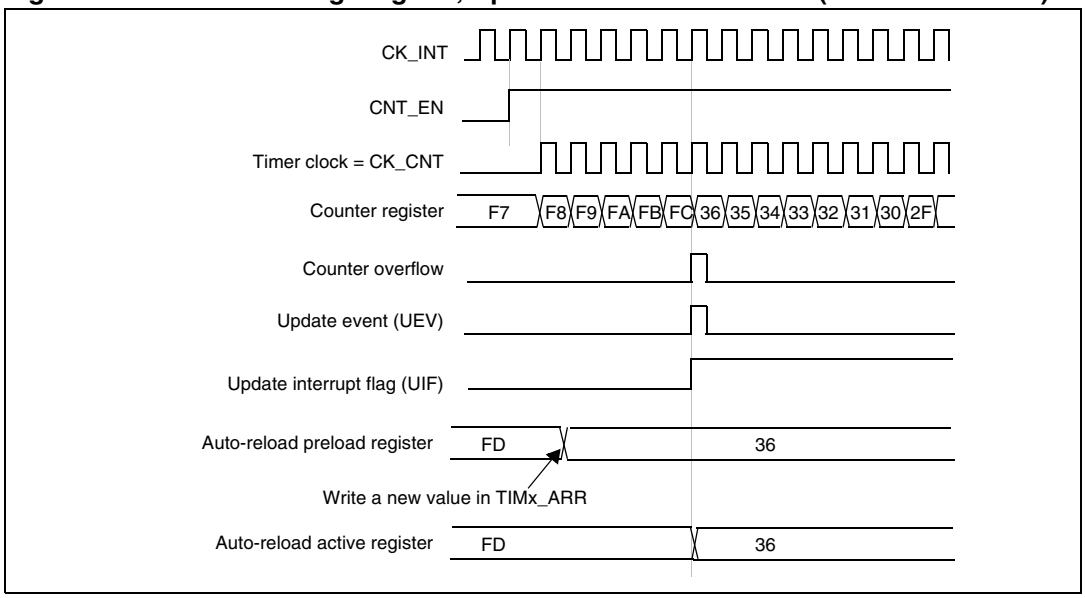


Figure 113. Counter timing diagram, Update event with ARPE=1 (counter overflow)



13.3.3 Clock selection

The counter clock can be provided by the following clock sources:

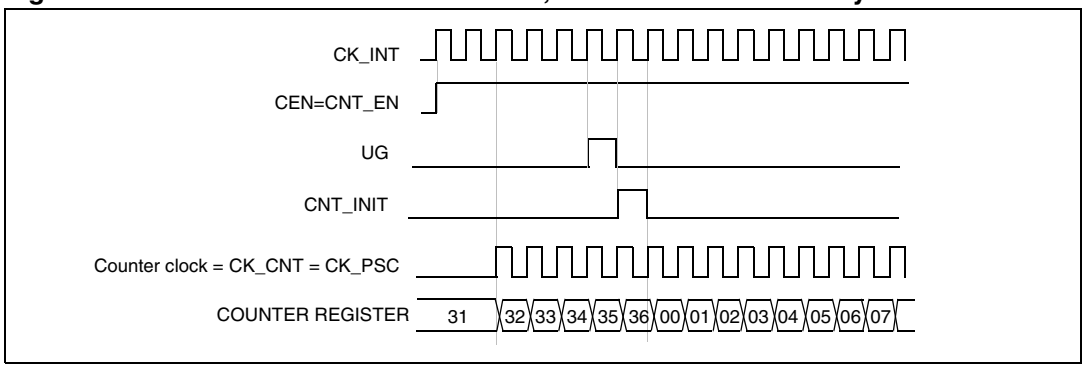
- Internal clock (CK_INT)
- External clock mode1: external input pin (TIx)
- External clock mode2: external trigger input (ETR)
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to *: Using one timer as prescaler for the another on page 297* for more details.

Internal clock source (CK_INT)

If the slave mode controller is disabled (SMS=000 in the TIMx_SMCR register), then the CEN, DIR (in the TIMx_CR1 register) and UG bits (in the TIMx_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK_INT.

Figure 114 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

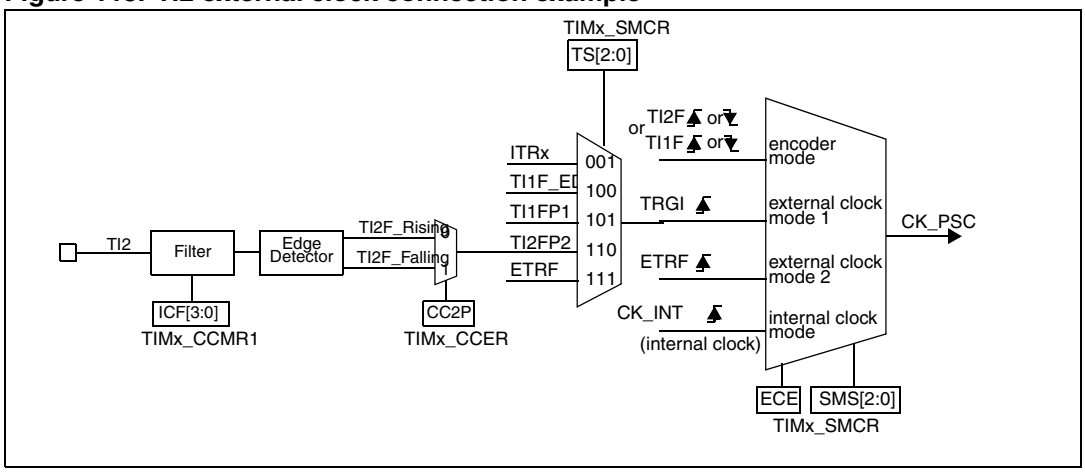
Figure 114. Control circuit in normal mode, internal clock divided by 1



External clock source mode 1

This mode is selected when SMS=111 in the TIMx_SMCR register. The counter can count at each rising or falling edge on a selected input.

Figure 115. TI2 external clock connection example



For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S= ‘01’ in the TIMx_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC2F=0000).

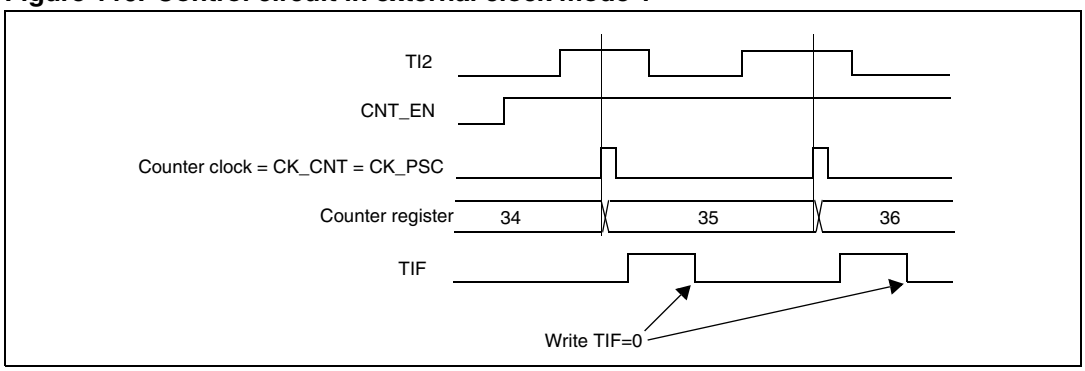
Note: The capture prescaler is not used for triggering, so you don’t need to configure it.

3. Select rising edge polarity by writing CC2P=0 in the TIMx_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx_SMCR register.
5. Select TI2 as the input source by writing TS=110 in the TIMx_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

The delay between the rising edge on TI2 and the actual clock of the counter is due to the resynchronization circuit on TI2 input.

Figure 116. Control circuit in external clock mode 1



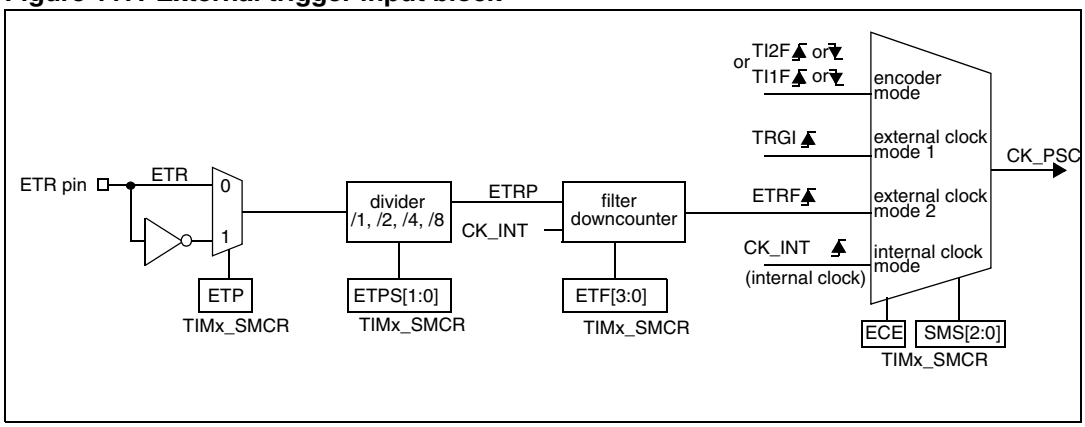
External clock source mode 2

This mode is selected by writing ECE=1 in the TIMx_SMCR register.

The counter can count at each rising or falling edge on the external trigger input ETR.

The *Figure 117* gives an overview of the external trigger input block.

Figure 117. External trigger input block



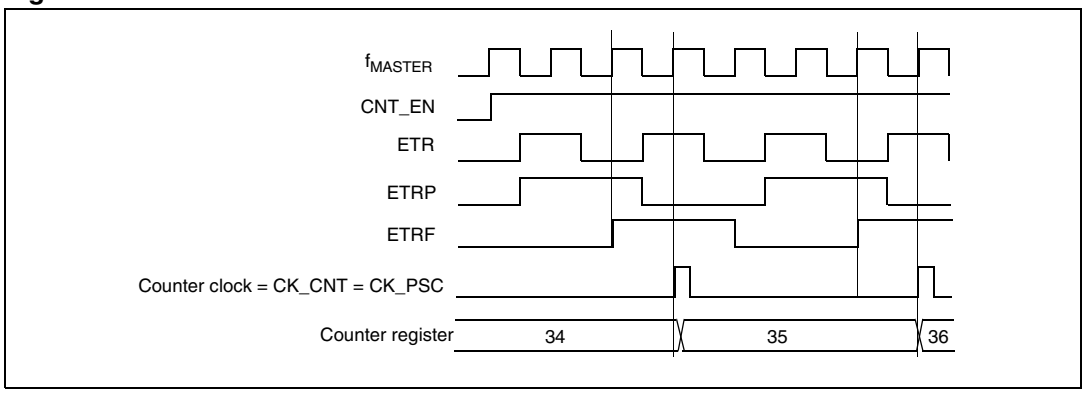
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0000 in the TIMx_SMCR register.
2. Set the prescaler by writing ETPS[1:0]=01 in the TIMx_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx_SMCR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR register.
5. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

The counter counts once each 2 ETR rising edges.

The delay between the rising edge on ETR and the actual clock of the counter is due to the resynchronization circuit on the ETRP signal.

Figure 118. Control circuit in external clock mode 2



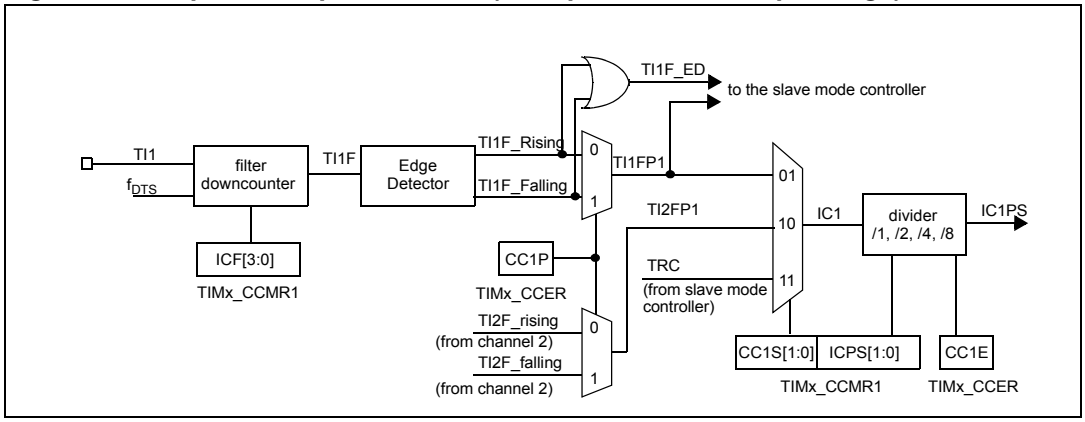
13.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

The following figure gives an overview of one Capture/Compare channel.

The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

Figure 119. Capture/compare channel (example: channel 1 input stage)



The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 120. Capture/compare channel 1 main circuit

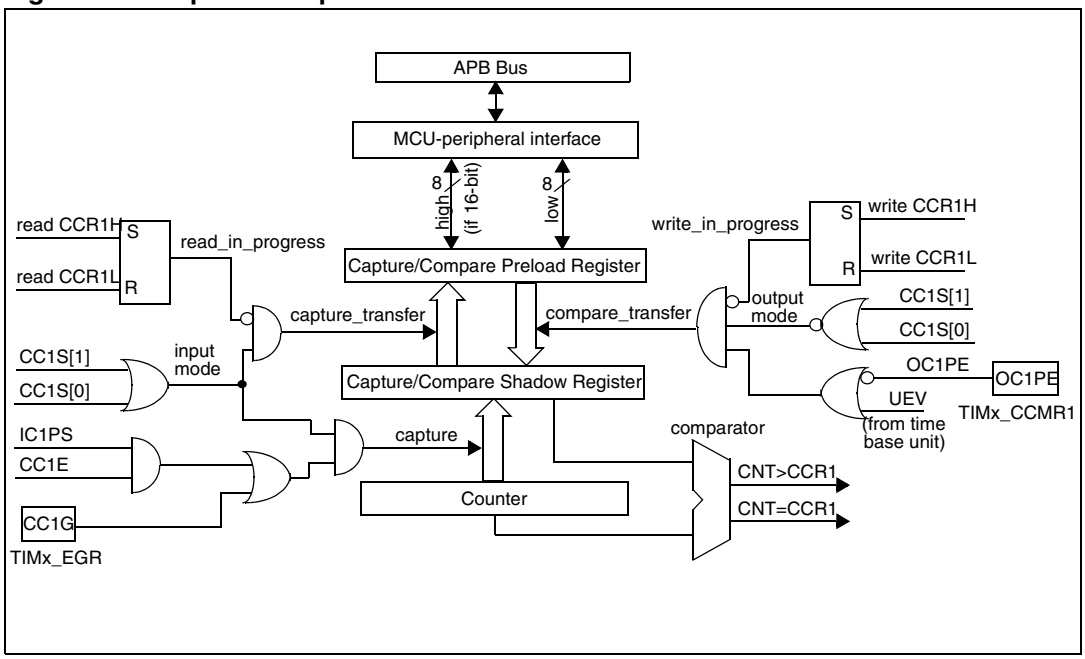
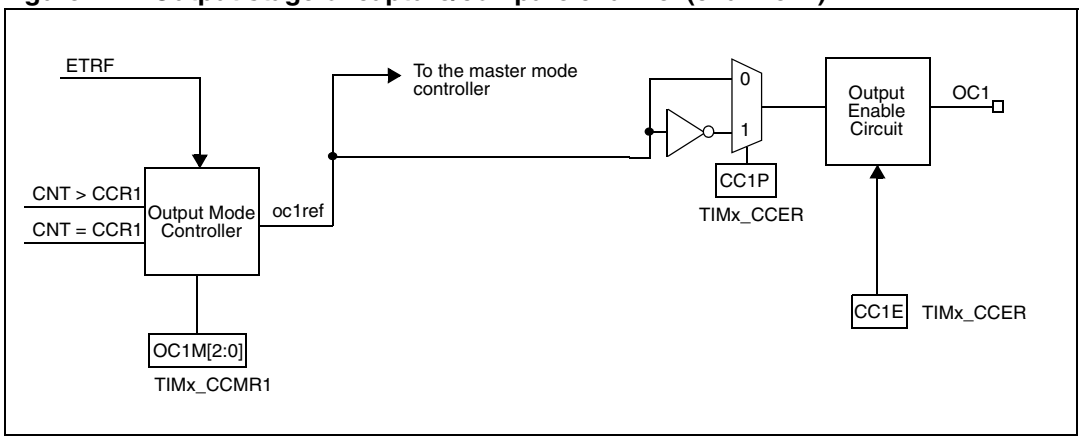


Figure 121. Output stage of capture/compare channel (channel 1)



The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

13.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCXIF flag (TIMx_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx_CCRx register. CCxOF is cleared when you write it to '0'.

The following example shows how to capture the counter value in TIMx_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx_CCR1 register becomes read-only.
- Program the input filter duration you need with respect to the signal you connect to the timer (when the input is one of the TIx (ICxF bits in the TIMx_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at must 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been detected (sampled at f_{DTS} frequency). Then write IC1F bits to 0011 in the TIMx_CCMR1 register.
- Select the edge of the active transition on the TI1 channel by writing CC1P bit to 0 in the TIMx_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 00 in the TIMx_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx_DIER register.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

Note: IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx_EGR register.

13.3.6 PWM input mode

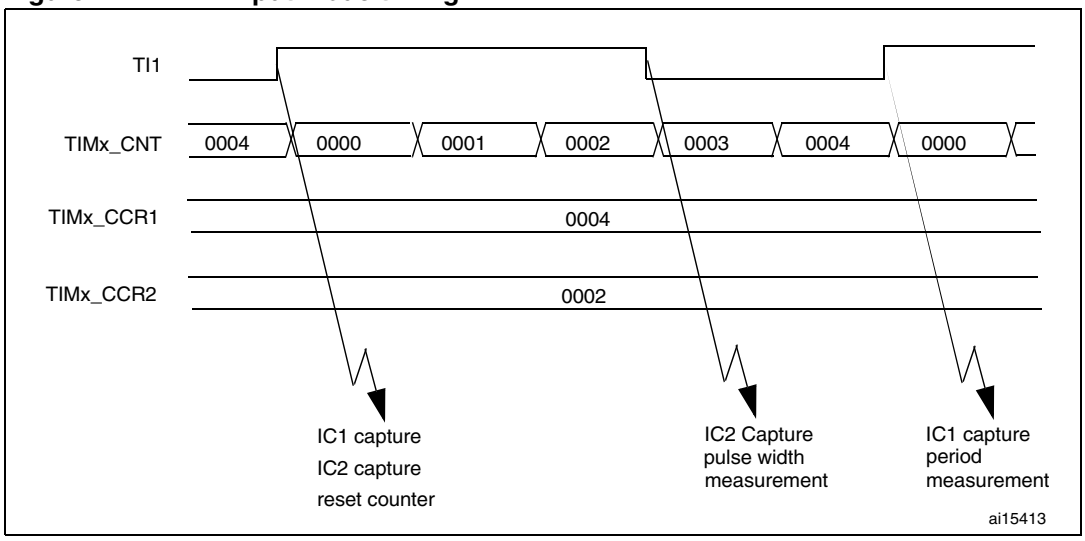
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, you can measure the period (in TIMx_CCR1 register) and the duty cycle (in TIMx_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK_INT frequency and prescaler value):

- Select the active input for TIMx_CCR1: write the CC1S bits to 01 in the TIMx_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): write the CC1P bit to '0' (active on rising edge).
- Select the active input for TIMx_CCR2: write the CC2S bits to 10 in the TIMx_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P bit to '1' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER register.

Figure 122. PWM input mode timing



1. The PWM input mode can be used only with the TIMx_CH1/TIMx_CH2 signals due to the fact that only TI1FP1 and TI2FP2 are connected to the slave mode controller.

13.3.7 Forced output mode

In output mode (CCxS bits = 00 in the TIMx_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (ocxref/OCx) to its active level, you just need to write 101 in the OCxM bits in the corresponding TIMx_CCMRx register. Thus ocxref is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

e.g.: CCxP=0 (OCx active high) => OCx is forced to high level.

ocxref signal can be forced low by writing the OCxM bits to 100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the Output Compare Mode section.

13.3.8 Output compare mode

This function is used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the capture/compare register and the counter, the output compare function:

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx_CCMRx register) and the output polarity (CCxP bit in the TIMx_CCER register). The output pin can keep its level (OCXM=000), be set active (OCxM=001), be set inactive (OCxM=010) or can toggle (OCxM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx_DIER register, CCDS bit in the TIMx_CR2 register for the DMA request selection).

The TIMx_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx_CCMRx register.

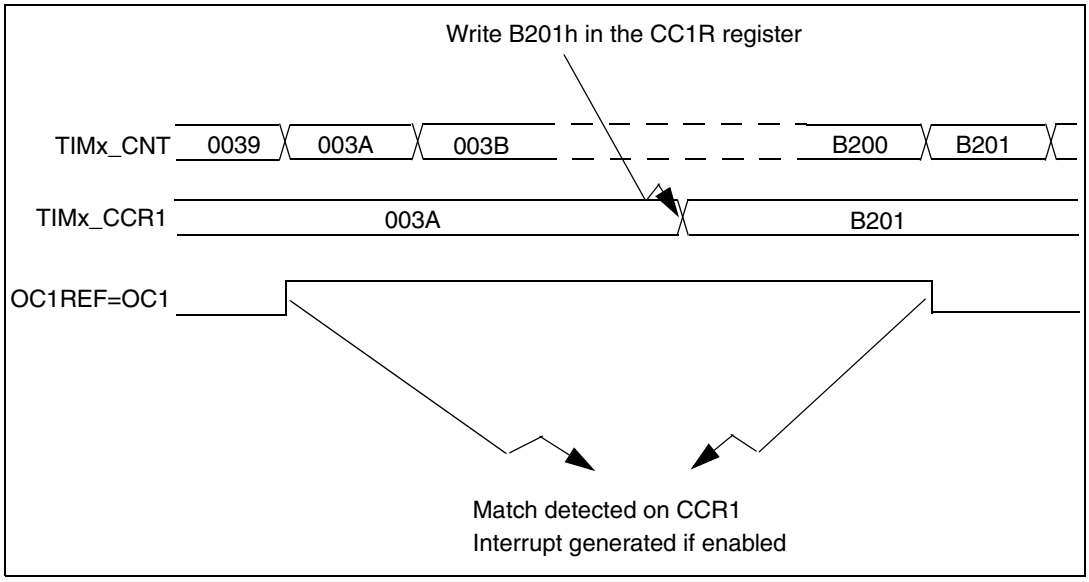
In output compare mode, the update event UEV has no effect on ocxref and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One Pulse Mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCRx registers.
3. Set the CCxIE and/or CCxDE bits if an interrupt and/or a DMA request is to be generated.
4. Select the output mode. For example, you must write OCxM=’011’, OCxPE=’0’, CCxP=’0’ and CCxE=’1’ to toggle OCx output pin when CNT matches CCRx, CCRx preload is not used, OCx is enabled and active high.
5. Enable the counter by setting the CEN bit in the TIMx_CR1 register.

The TIMx_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE=’0’, else TIMx_CCRx shadow register is updated only at the next update event UEV). An example is given in *Figure 123*.

Figure 123. Output compare mode, toggle on OC1.



13.3.9 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing ‘110’ (PWM mode 1) or ‘111’ (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by the CCxE bit in the TIMx_CCER register. Refer to the TIMx_CCERx register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx≤TIMx_CNT or TIMx_CNT≤TIMx_CCRx (depending on the direction of the counter). However, to comply with the OCREF_CLR functionality (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison changes, or
- When the output compare mode (OCxM bits in TIMx_CCMRx register) switches from the “frozen” configuration (no comparison, OCxM=‘000’) to one of the PWM modes (OCxM=‘110’ or ‘111’).

This allows to force the PWM by software while running.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

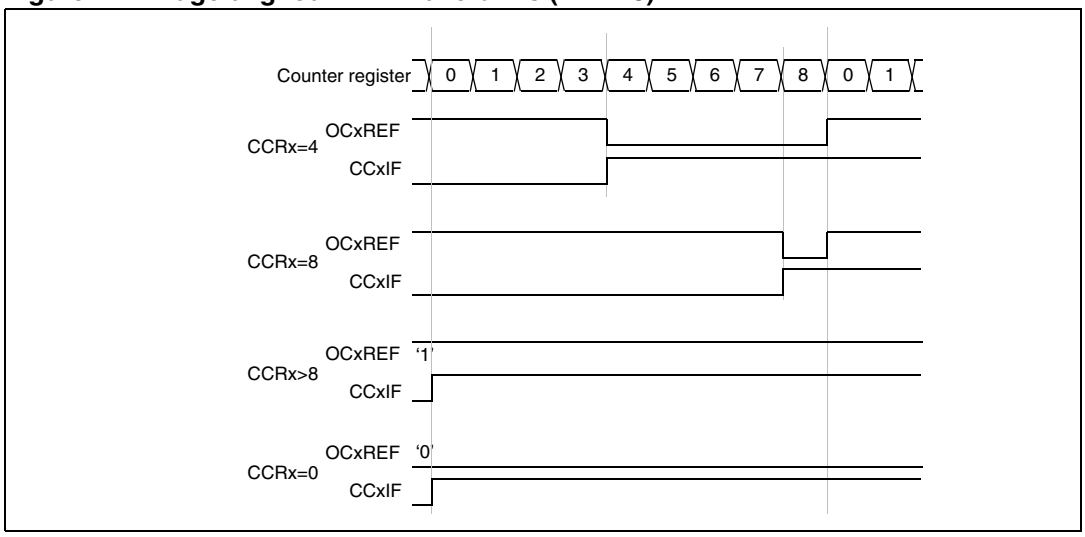
PWM edge-aligned mode

Upcounting configuration

Upcounting is active when the DIR bit in the TIMx_CR1 register is low. Refer to the *Section : upcounting mode on page 270*.

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as TIMx_CNT <TIMx_CCRx else it becomes low. If the compare value in TIMx_CCRx is greater than the auto-reload value (in TIMx_ARR) then OCxREF is held at ‘1’. If the compare value is 0 then OCxREF is held at ‘0’. *Figure 124* shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

Figure 124. Edge-aligned PWM waveforms (ARR=8)



Downcounting configuration

Downcounting is active when DIR bit in TIMx_CR1 register is high. Refer to *Downcounting mode on page 273*

In PWM mode 1, the reference signal ocxref is low as long as TIMx_CNT>TIMx_CCRx else it becomes high. If the compare value in TIMx_CCRx is greater than the auto-reload value in TIMx_ARR, then ocxref is held at ‘1’. 0% PWM is not possible in this mode.

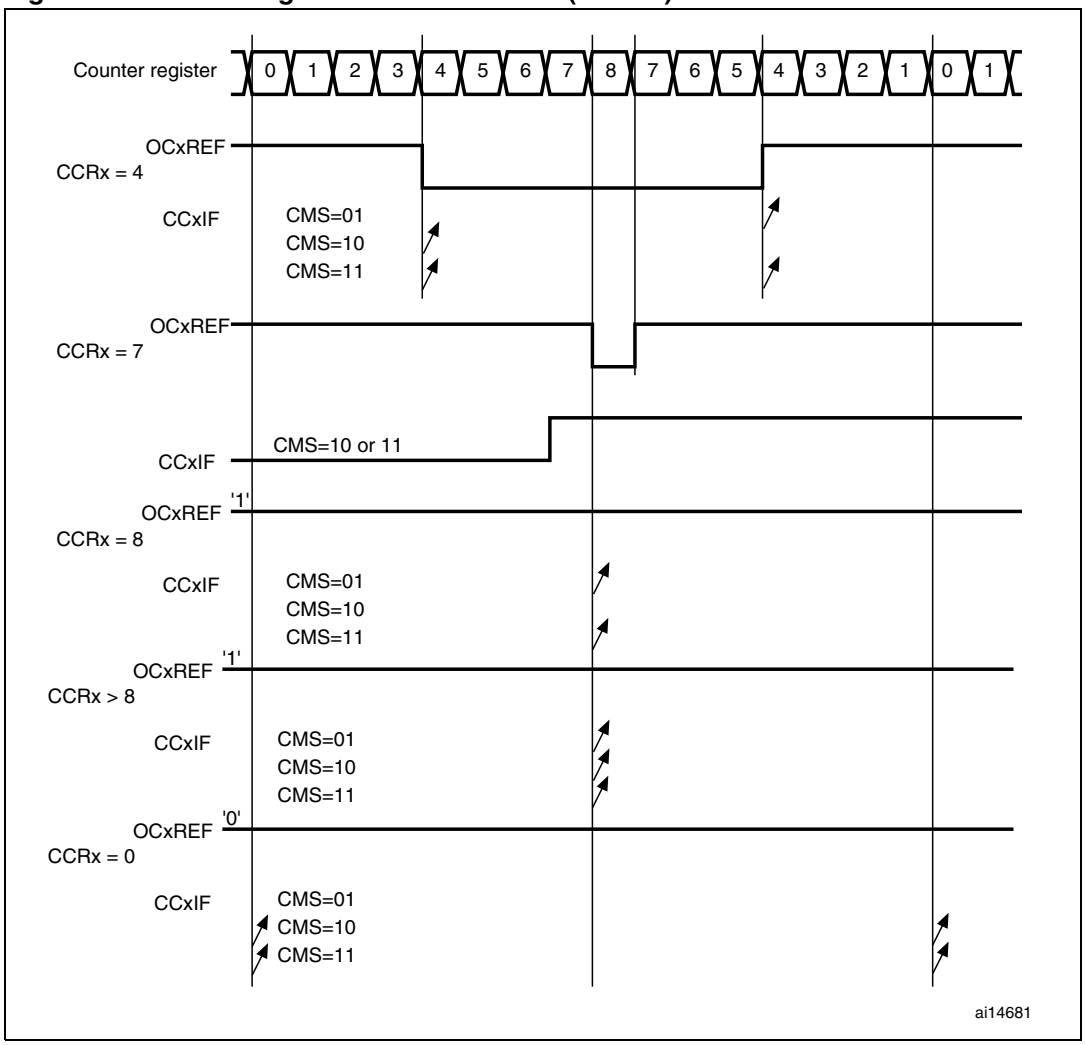
PWM center-aligned mode

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are different from ‘00’ (all the remaining configurations having the same effect on the ocxref/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx_CR1 register is updated by hardware and must not be changed by software. Refer to the *Center-aligned mode (up/down counting) on page 275*.

Figure 125 shows some center-aligned PWM waveforms in an example where:

- TIMx_ARR=8,
- PWM mode is the PWM mode 1,
- The flag is set when the counter counts down corresponding to the center-aligned mode 1 selected for CMS=01 in TIMx_CR1 register.

Figure 125. Center-aligned PWM waveforms (ARR=8)



Hints on using center-aligned mode:

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter counts up or down depending on the value written in the DIR bit in the TIMx_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if you write a value in the counter that is greater than the auto-reload value (TIMx_CNT>TIMx_ARR). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if you write 0 or write the TIMx_ARR value in the counter but no Update Event UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIMx_EGR register) just before starting the counter and not to write the counter while it is running.

13.3.10 One pulse mode

One Pulse Mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

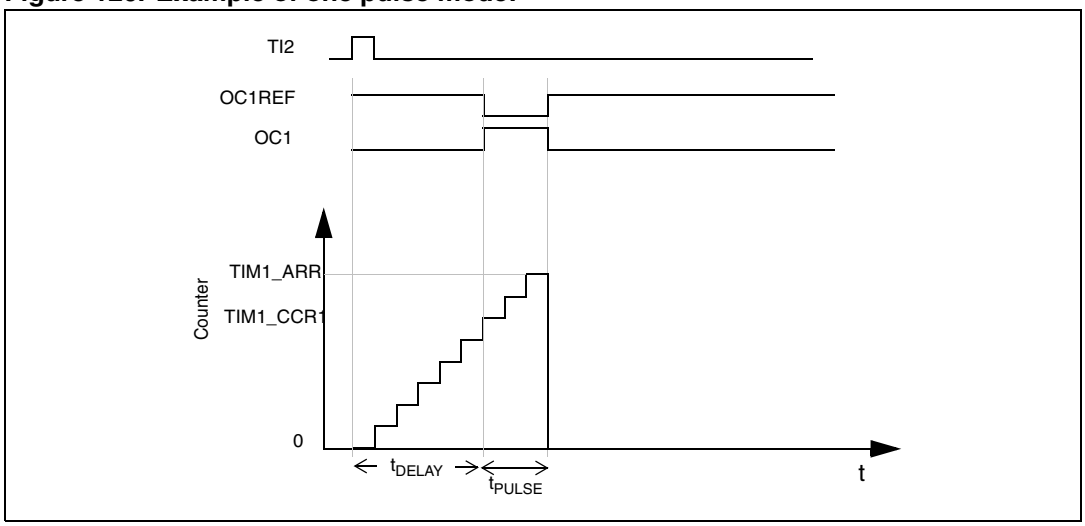
Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. You select One Pulse Mode by setting the OPM bit in the TIMx_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

In upcounting: CNT<CCRx≤ARR (in particular, 0<CCRx),

In downcounting: CNT>CCRx.

Figure 126. Example of one pulse mode.



For example you may want to generate a positive pulse on OC1 with a length of t_{PULSE} and after a delay of t_{DELAY} as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 on TI2 by writing IC2S='01' in the TIMx_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' in the TIMx_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIMx_SMCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

- The t_{DELAY} is defined by the value written in the TIMx_CCR1 register.
- The t_{PULSE} is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1).
- Let's say you want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this you enable PWM mode 2 by writing OC1M=111 in the TIMx_CCMR1 register. You can optionally enable the preload registers by writing OC1PE='1' in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case you have to write the compare value in the TIMx_CCR1 register, the auto-reload value in the TIMx_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx_CR1 register should be low.

You only want 1 pulse, so you write '1' in the OPM bit in the TIMx_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

Particular case: OCx fast enable:

In One Pulse Mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay t_{DELAY} min we can get.

If you want to output a waveform with the minimum delay, you can set the OCxFE bit in the TIMx_CCMRx register. Then OCxRef (and OCx) is forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

13.3.11 Clearing the OCxREF signal on an external event

The OCxREF signal for a given channel can be reset by applying a High level on the ETRF input (OCxCE enable bit of the corresponding TIMx_CCMRx register set to '1'). The OCxREF remains low until the next update event, UEV, occurs.

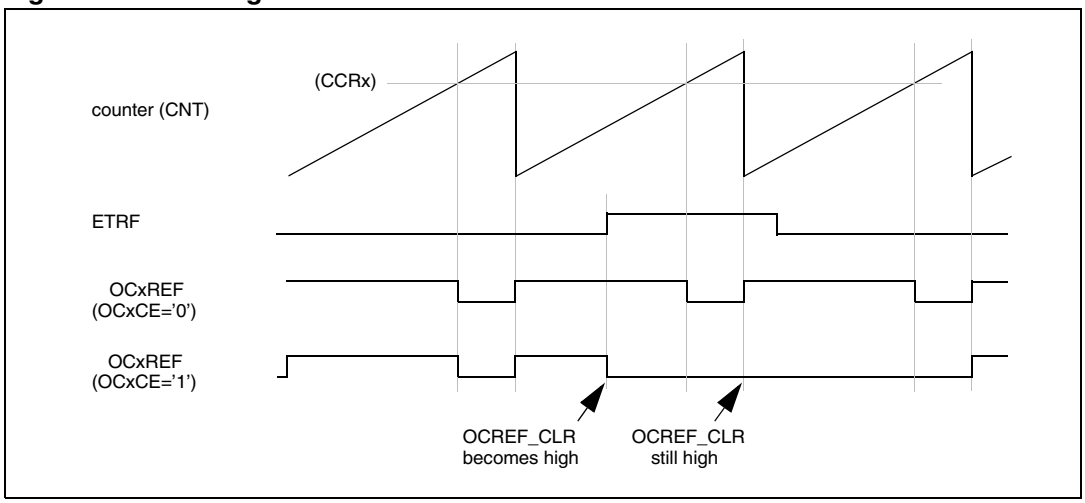
This function can be only used in output compare mode and PWM mode. It does not work in forced mode.

For example, the OCxREF signal can be connected to the output of a comparator to be used for current handling. In this case, the ETR must be configured as follow:

1. The external trigger prescaler should be kept off: bits ETPS[1:0] of the TIMx_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIM1_SMCR register set to '0'.
3. The external trigger polarity (ETP) and the external trigger filter (ETF) can be configured according to the user needs.

Figure 127 shows the behavior of the OCxREF signal when the ETRF Input becomes High, for both values of the enable bit OCxCE. In this example, the timer TIMx is programmed in PWM mode.

Figure 127. Clearing TIMx OCxREF



13.3.12 Encoder interface mode

To select Encoder Interface mode write SMS='001' in the TIMx_SMCR register if the counter is counting on TI2 edges only, SMS='010' if it is counting on TI1 edges only and SMS='011' if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx_CCER register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to *Table 58*. The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIMx_ARR before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

Table 58. Counting direction versus encoder signals

Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No Count	No Count
	Low	Up	Down	No Count	No Count
Counting on TI2 only	High	No Count	No Count	Up	Down
	Low	No Count	No Count	Down	Up
Counting on TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators are normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

The *Figure 128* gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S='01' (TIMx_CCMR1 register, IC1FP1 mapped on TI1).
- CC2S='01' (TIMx_CCMR2 register, IC2FP2 mapped on TI2).
- CC1P='0' (TIMx_CCER register, IC1FP1 non-inverted, IC1FP1=TI1).
- CC2P='0' (TIMx_CCER register, IC2FP2 non-inverted, IC2FP2=TI2).
- SMS='011' (TIMx_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIMx_CR1 register, Counter is enabled).

Figure 128. Example of counter operation in encoder interface mode.

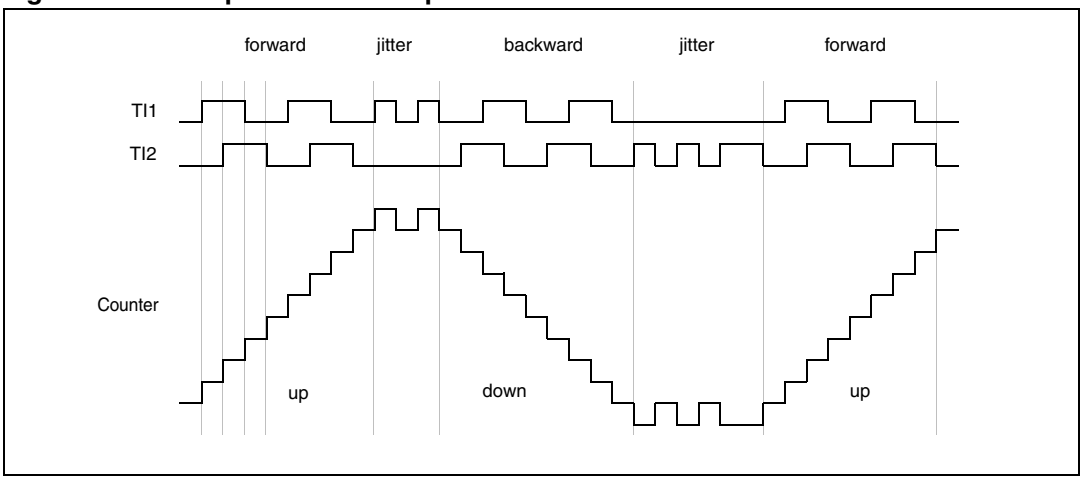
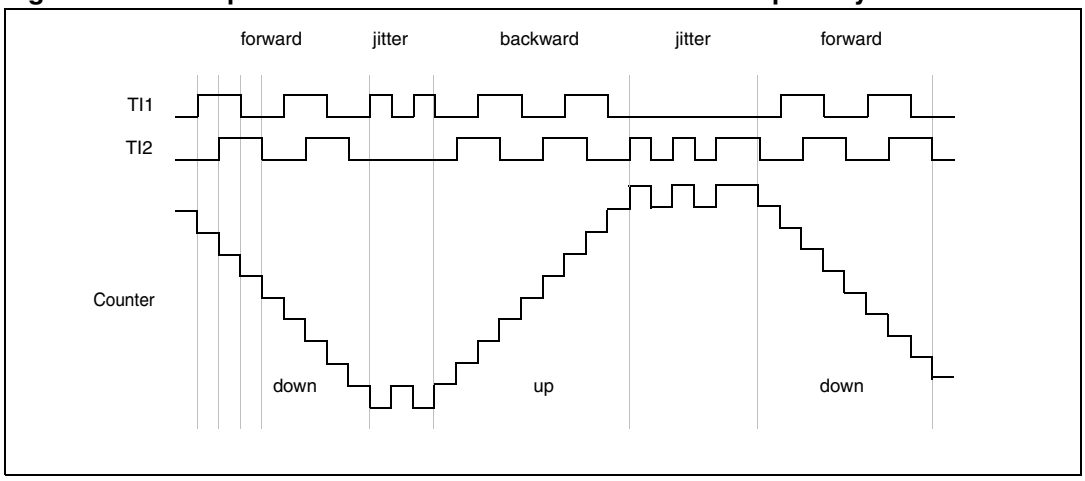


Figure 129 gives an example of counter behavior when IC1FP1 polarity is inverted (same configuration as above except CC1P='1').

Figure 129. Example of encoder interface mode with IC1FP1 polarity inverted.



The timer, when configured in Encoder Interface mode provides information on the sensor's current position. You can obtain dynamic information (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. You can do this by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). when available, it is also possible to read its value through a DMA request generated by a Real-Time clock.

13.3.13 Timer input XOR function

The TI1S bit in the TIM1_CR2 register, allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx_CH1 to TIMx_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture.

An example of this feature used to interface Hall sensors is given in *Section 12.3.18 on page 234*.

13.3.14 Timers and external trigger synchronization

The TIMx Timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx_ARR, TIMx_CCRx) are updated.

In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

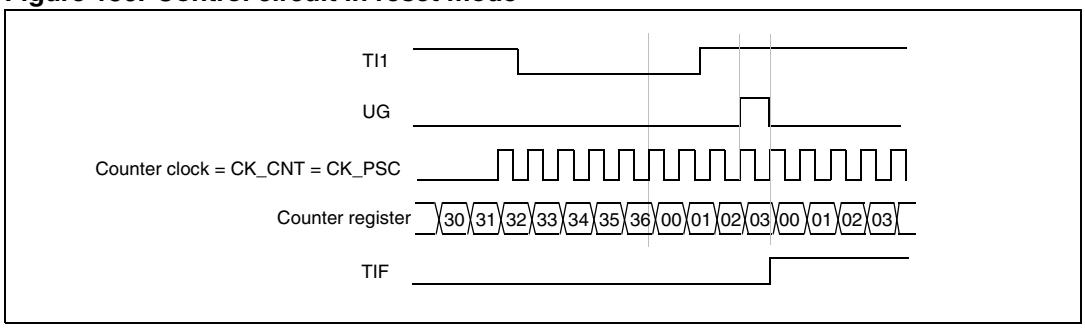
- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx_CCMR1 register. Write CC1P=0 in TIMx_CCER register to validate the polarity (and detect rising edges only).

- Configure the timer in reset mode by writing SMS=100 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Start the counter by writing CEN=1 in the TIMx_CR1 register.

The counter starts counting on the internal clock, then behaves normally until TI1 rising edge. When TI1 rises, the counter is cleared and restarts from 0. In the meantime, the trigger flag is set (TIF bit in the TIMx_SR register) and an interrupt request, or a DMA request can be sent if enabled (depending on the TIE and TDE bits in TIMx_DIER register).

The following figure shows this behavior when the auto-reload register TIMx_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

Figure 130. Control circuit in reset mode



Slave mode: Gated mode

The counter can be enabled depending on the level of a selected input.

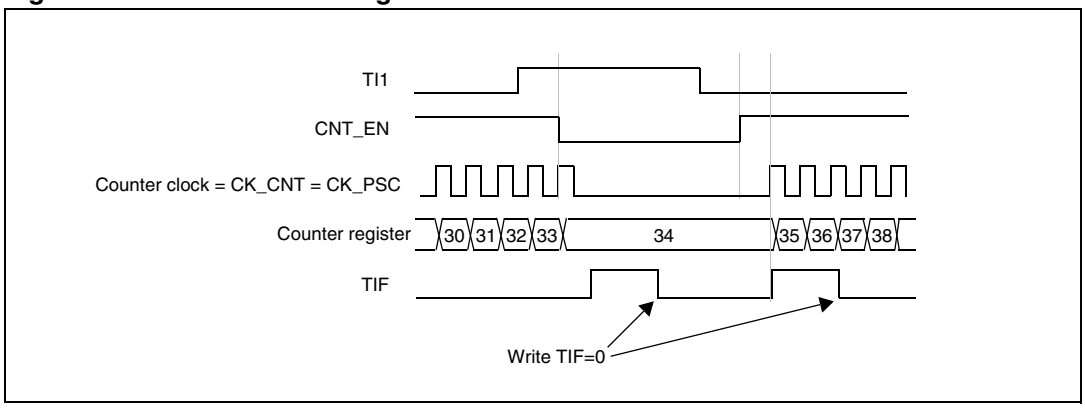
In the following example, the upcounter counts only when TI1 input is low:

- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S=01 in TIMx_CCMR1 register. Write CC1P=1 in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

The counter starts counting on the internal clock as long as TI1 is low and stops as soon as TI1 becomes high. The TIF flag in the TIMx_SR register is set both when the counter starts or stops.

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

Figure 131. Control circuit in gated mode



Slave mode: Trigger mode

The counter can start in response to an event on a selected input.

In the following example, the upcounter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. CC2S bits are selecting the input capture source only, CC2s=01 in TIMx_CCMR1 register. Write CC2P=1 in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx_SMCR register.

When a rising edge occurs on TI2, the counter starts counting on the internal clock and the TIF flag is set.

The delay between the rising edge on TI2 and the actual start of the counter is due to the resynchronization circuit on TI2 input.

Figure 132. Control circuit in trigger mode

Slave mode: External Clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input when operating in reset mode, gated mode or trigger mode. It is recommended not to select ETR as TRGI through the TS bits of TIMx_SMCR register.

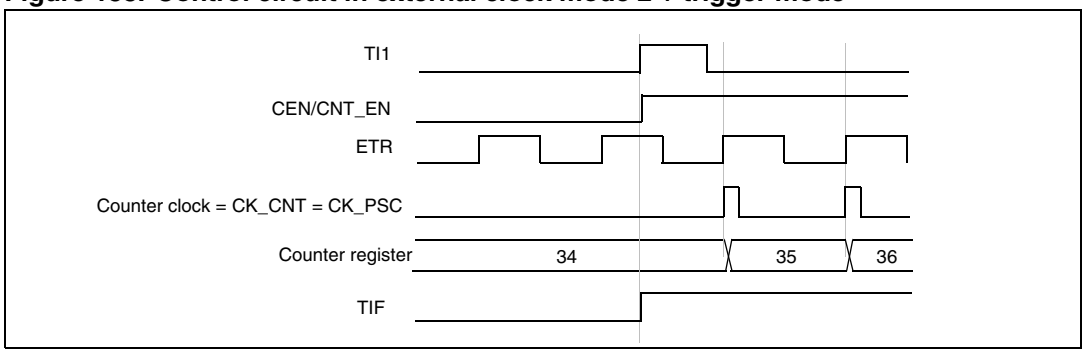
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

1. Configure the external trigger input circuit by programming the TIMx_SMCR register as follows:
   - ETF = 0000: no filter
   - ETPS=00: prescaler disabled
   - ETP=0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.
2. Configure the channel 1 as follows, to detect rising edges on TI:
   - IC1F=0000: no filter.
   - The capture prescaler is not used for triggering and does not need to be configured.
   - CC1S=01in TIMx_CCMR1 register to select only the input capture source
   - CC1P=0 in TIMx_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.

A rising edge on TI1 enables the counter and sets the TIF flag. The counter then counts on ETR rising edges.

The delay between the rising edge of the ETR signal and the actual reset of the counter is due to the resynchronization circuit on ETRP input.

Figure 133. Control circuit in external clock mode 2 + trigger mode



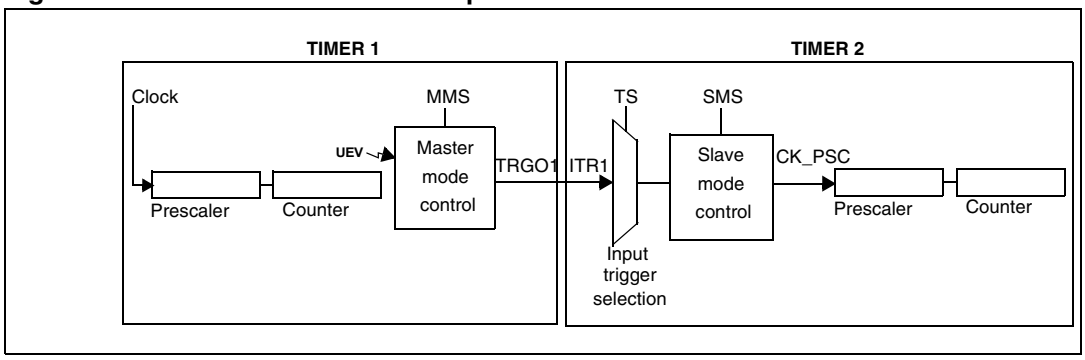
13.3.15 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.

The following figure presents an overview of the trigger selection and the master mode selection blocks.

Using one timer as prescaler for the another

Figure 134. Master/Slave timer example



For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to *Figure 134*. To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each update event UEV. If you write MMS=010 in the TIM1_CR2 register, a rising edge is output on TRGO1 each time an update event is generated.
- To connect the TRGO1 output of Timer 1 to Timer 2, Timer 2 must be configured in slave mode using ITR1 as internal trigger. You select this through the TS bits in the TIM2_SMCR register (writing TS=000).
- Then you put the slave mode controller in external clock mode 1 (write SMS=111 in the TIM2_SMCR register). This causes Timer 2 to be clocked by the rising edge of the periodic Timer 1 trigger signal (which correspond to the timer 1 counter overflow).
- Finally both timers must be enabled by setting their respective CEN bits (TIMx_CR1 register).

Note: If OCx is selected on Timer 1 as trigger output (MMS=1xx), its rising edge is used to clock the counter of timer 2.

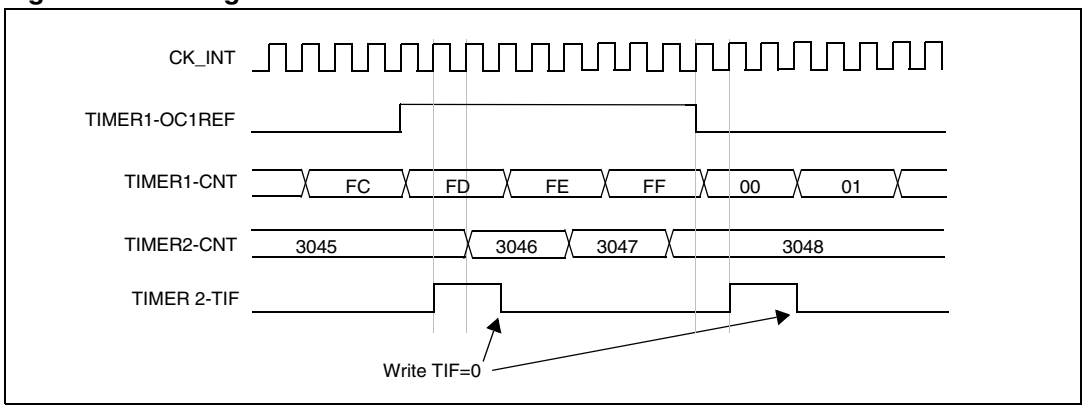
Using one timer to enable another timer

In this example, we control the enable of Timer 2 with the output compare 1 of Timer 1. Refer to *Figure 134* for connections. Timer 2 counts on the divided internal clock only when OC1REF of Timer 1 is high. Both counter clock frequencies are divided by 3 by the prescaler compared to CK_INT ($f_{CK_CNT} = f_{CK_INT}/3$).

- Configure Timer 1 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM1_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2_SMCR register).
- Enable Timer 2 by writing ‘1’ in the CEN bit (TIM2_CR1 register).
- Start Timer 1 by writing ‘1’ in the CEN bit (TIM1_CR1 register).

Note: The counter 2 clock is not synchronized with counter 1, this mode only affects the Timer 2 counter enable signal.

Figure 135. Gating timer 2 with OC1REF of timer 1

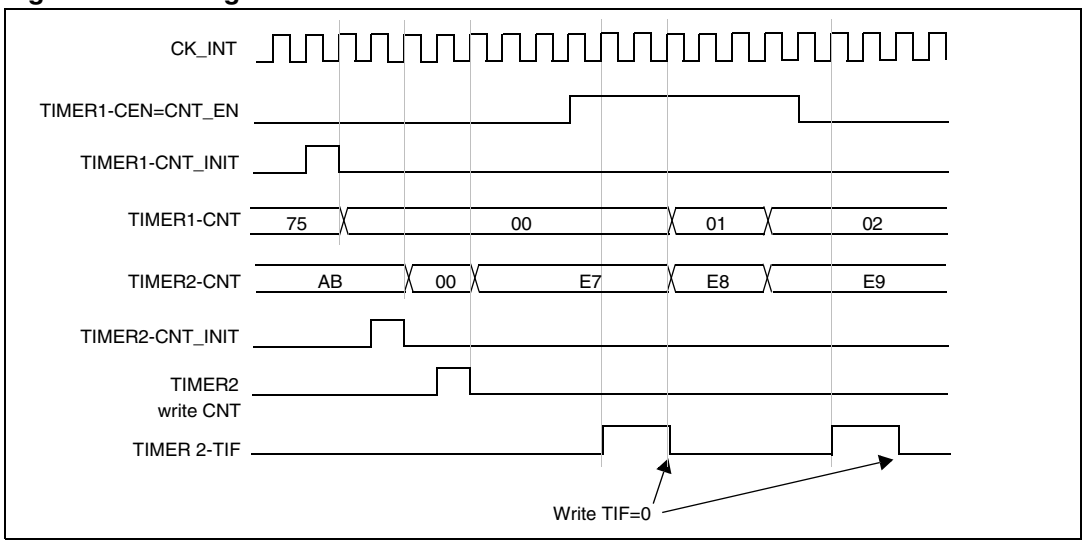


In the example in *Figure 135*, the Timer 2 counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer 1. You can then write any value you want in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx_EGR registers.

In the next example, we synchronize Timer 1 and Timer 2. Timer 1 is the master and starts from 0. Timer 2 is the slave and starts from 0xE7. The prescaler ratio is the same for both timers. Timer 2 stops when Timer 1 is disabled by writing ‘0’ to the CEN bit in the TIM1_CR1 register:

- Configure Timer 1 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM1_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2_SMCR register).
- Reset Timer 1 by writing ‘1’ in UG bit (TIM1_EGR register).
- Reset Timer 2 by writing ‘1’ in UG bit (TIM2_EGR register).
- Initialize Timer 2 to 0xE7 by writing ‘0xE7’ in the timer 2 counter (TIM2_CNTL).
- Enable Timer 2 by writing ‘1’ in the CEN bit (TIM2_CR1 register).
- Start Timer 1 by writing ‘1’ in the CEN bit (TIM1_CR1 register).
- Stop Timer 1 by writing ‘0’ in the CEN bit (TIM1_CR1 register).

Figure 136. Gating timer 2 with Enable of timer 1

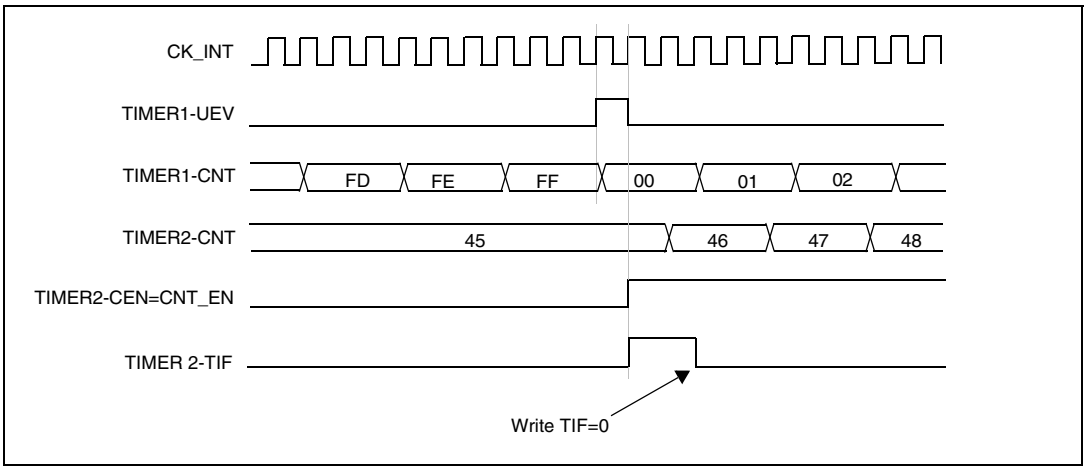


Using one timer to start another timer

In this example, we set the enable of Timer 2 with the update event of Timer 1. Refer to *Figure 134* for connections. Timer 2 starts counting from its current value (which can be non-zero) on the divided internal clock as soon as the update event is generated by Timer 1. When Timer 2 receives the trigger signal its CEN bit is automatically set and the counter counts until we write ‘0’ to the CEN bit in the TIM2_CR1 register. Both counter clock frequencies are divided by 3 by the prescaler compared to CK_INT ($f_{CK_CNT} = f_{CK_INT}/3$).

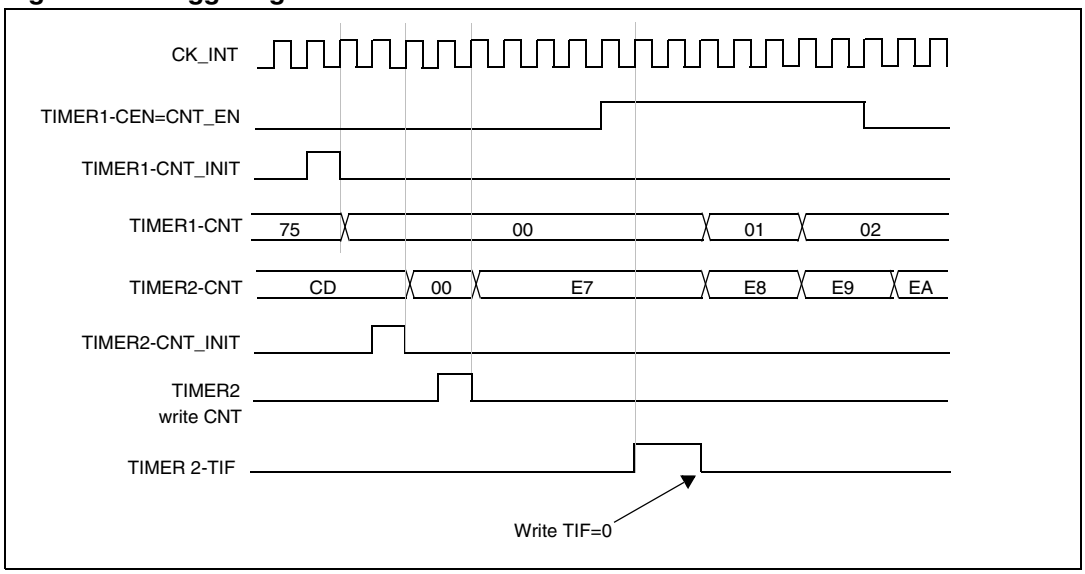
- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1_CR2 register).
- Configure the Timer 1 period (TIM1_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2_SMCR register).
- Configure Timer 2 in trigger mode (SMS=110 in TIM2_SMCR register).
- Start Timer 1 by writing ‘1’ in the CEN bit (TIM1_CR1 register).

Figure 137. Triggering timer 2 with Update of timer 1



As in the previous example, you can initialize both counters before starting counting. *Figure 138* shows the behavior with the same configuration as in *Figure 137* but in trigger mode instead of gated mode (SMS=110 in the TIM2_SMCR register).

Figure 138. Triggering timer 2 with Enable of timer 1



Using one timer as prescaler for another timer

For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to *Figure 134* for connections. To do this:

- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1_CR2 register). then it outputs a periodic signal on each counter overflow.
- Configure the Timer 1 period (TIM1_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2_SMCR register).
- Configure Timer 2 in external clock mode 1 (SMS=111 in TIM2_SMCR register).
- Start Timer 2 by writing ‘1’ in the CEN bit (TIM2_CR1 register).
- Start Timer 1 by writing ‘1’ in the CEN bit (TIM1_CR1 register).

Starting 2 timers synchronously in response to an external trigger

In this example, we set the enable of timer 1 when its TI1 input rises, and the enable of Timer 2 with the enable of Timer 1. Refer to *Figure 134* for connections. To ensure the

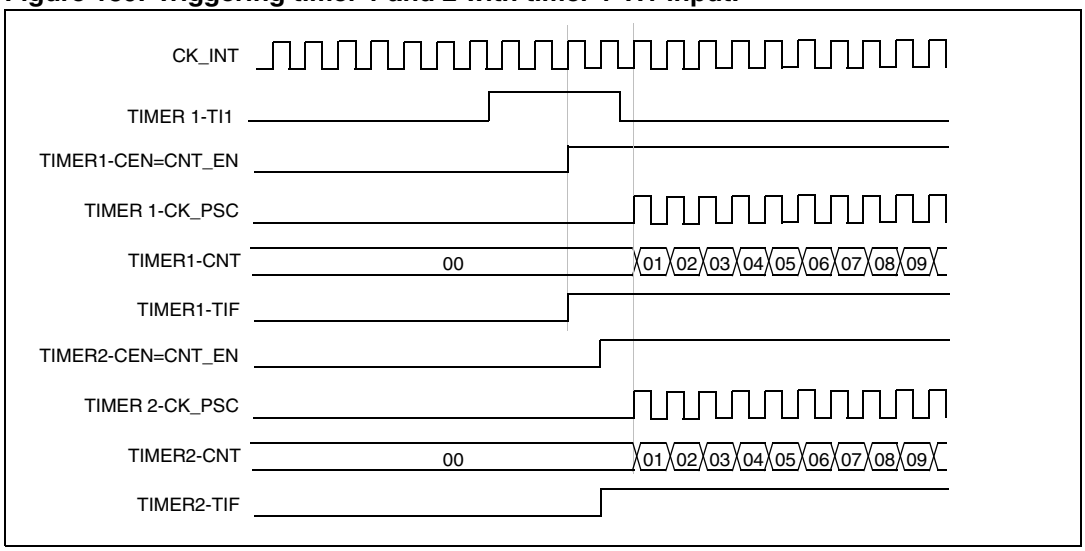
counters are aligned, Timer 1 must be configured in Master/Slave mode (slave with respect to TI1, master with respect to Timer 2):

- Configure Timer 1 master mode to send its Enable as trigger output (MMS=001 in the TIM1_CR2 register).
- Configure Timer 1 slave mode to get the input trigger from TI1 (TS=100 in the TIM1_SMCR register).
- Configure Timer 1 in trigger mode (SMS=110 in the TIM1_SMCR register).
- Configure the Timer 1 in Master/Slave mode by writing MSM='1' (TIM1_SMCR register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2_SMCR register).
- Configure Timer 2 in trigger mode (SMS=110 in the TIM2_SMCR register).

When a rising edge occurs on TI1 (Timer 1), both counters starts counting synchronously on the internal clock and both TIF flags are set.

Note: In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but you can easily insert an offset between them by writing any of the counter registers (TIMx_CNT). You can see that the master/slave mode insert a delay between CNT_EN and CK_PSC on timer 1.

Figure 139. Triggering timer 1 and 2 with timer 1 TI1 input.



13.3.16 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core - halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module. For more details, refer to *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C*.

13.4 TIMx registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

13.4.1 Control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved, always read as 0

Bits 9:8 **CKD**: *Clock division.*

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

00: $t_{DTS} = t_{CK_INT}$
01: $t_{DTS} = 2 \times t_{CK_INT}$
10: $t_{DTS} = 4 \times t_{CK_INT}$
11: Reserved

Bit 7 **ARPE**: *Auto-Reload Preload enable.*

0: TIMx_ARR register is not buffered.
1: TIMx_ARR register is buffered.

Bits 6:5 **CMS**: *Center-aligned Mode Selection.*

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

Bit 4 **DIR**: *Direction.*

0: Counter used as upcounter.
1: Counter used as downcounter.

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: *One Pulse Mode.*

0: Counter is not stopped at update event
1: Counter stops counting at the next update event (clearing the bit CEN).

Bit 2 **URS**: *Update Request Source.*

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled. These events can be:

– Counter overflow/underflow
– Setting the UG bit
– Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: *Update DIsable.*

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

– Counter overflow/underflow
– Setting the UG bit
– Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: *Counter enable.*

0: Counter disabled
1: Counter enabled

Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one pulse mode, when an update event occurs.

13.4.2 Control register 2 (TIMx_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TI1S	MMS[2:0]			CCDS	Reserved		
								rw	rw	rw	rw	rw			

Bits 15:8 Reserved, always read as 0.

Bit 7 **TI1S**: *TI1 Selection.*

0: The TIMx_CH1 pin is connected to TI1 input.
1: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

See also *Section 12.3.18: Interfacing with Hall sensors on page 234*

Bits 6:4 **MMS**: *Master Mode Selection.*

These bits allow to select the information to be sent in master mode to slave timers for synchronization (TRGO). The combination is as follows:
000: **Reset** - the UG bit from the TIMx_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT_EN, is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.
011: **Compare Pulse** - The trigger output send a positive pulse when the CC1IF flag is to be set (even if it was already high), as soon as a capture or a compare match occurred. (TRGO).
100: **Compare** - OC1REF signal is used as trigger output (TRGO).
101: **Compare** - OC2REF signal is used as trigger output (TRGO).
110: **Compare** - OC3REF signal is used as trigger output (TRGO).
111: **Compare** - OC4REF signal is used as trigger output (TRGO).

Bit 3 **CCDS**: *Capture/Compare DMA Selection.*

0: CCx DMA request sent when CCx event occurs
1: CCx DMA requests sent when update event occurs

Bits 2:0 Reserved, always read as 0

13.4.3 Slave mode control register (TIMx_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

Bit 15 **ETP**: *External Trigger Polarity.*

This bit selects whether ETR or $\overline{ETR}$ is used for trigger operations
0: ETR is non-inverted, active at high level or rising edge.
1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE**: *External Clock enable.*

This bit enables External clock mode 2.
0: External clock mode 2 disabled.
1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

Note 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).
Note 2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).
Note 3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS**: *External Trigger Prescaler.*

External trigger signal ETRP frequency must be at most 1/4 of CK_INT frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF.
01: ETRP frequency divided by 2.
10: ETRP frequency divided by 4.
11: ETRP frequency divided by 8.

Bits 11:8 **ETF[3:0]**: *External Trigger Filter.*

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS}.
0001: $f_{SAMPLING}=f_{CK_INT}$, N=2.
0010: $f_{SAMPLING}=f_{CK_INT}$, N=4.
0011: $f_{SAMPLING}=f_{CK_INT}$, N=8.
0100: $f_{SAMPLING}=f_{DTS}/2$, N=6.
0101: $f_{SAMPLING}=f_{DTS}/2$, N=8.
0110: $f_{SAMPLING}=f_{DTS}/4$, N=6.
0111: $f_{SAMPLING}=f_{DTS}/4$, N=8.
1000: $f_{SAMPLING}=f_{DTS}/8$, N=6.
1001: $f_{SAMPLING}=f_{DTS}/8$, N=8.
1010: $f_{SAMPLING}=f_{DTS}/16$, N=5.
1011: $f_{SAMPLING}=f_{DTS}/16$, N=6.
1100: $f_{SAMPLING}=f_{DTS}/16$, N=8.
1101: $f_{SAMPLING}=f_{DTS}/32$, N=5.
1110: $f_{SAMPLING}=f_{DTS}/32$, N=6.
1111: $f_{SAMPLING}=f_{DTS}/32$, N=8.

Bit 7 **MSM:** *Master/Slave mode.*

0: No action
1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS:** *Trigger Selection.*

This bit-field selects the trigger input to be used to synchronize the counter.
000: Internal Trigger 0 (ITR0). TIM1
001: Internal Trigger 1 (ITR1). TIM2
010: Internal Trigger 2 (ITR2). TIM3
011: Internal Trigger 3 (ITR3). TIM4
100: TI1 Edge Detector (TI1F_ED).
101: Filtered Timer Input 1 (TI1FP1).
110: Filtered Timer Input 2 (TI2FP2).
111: External Trigger input (ETRF).
See *Table 59: TIMx Internal trigger connection on page 306* for more details on ITRx meaning for each Timer.

Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.

Bit 3 Reserved, always read as 0.

Bits 2:0 **SMS:** *Slave Mode Selection.*

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description.
000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.
001: Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.
010: Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.
011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.
100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.
110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.
111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS='100'). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

Table 59. TIMx Internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1	TIM8	TIM3	TIM4
TIM3	TIM1	TIM2	TIM5	TIM4
TIM4	TIM1	TIM2	TIM3	TIM8
TIM5	TIM2	TIM3	TIM4	TIM8

13.4.4 DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4 DE	CC3 DE	CC2 DE	CC1 DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 15 Reserved, always read as 0.

Bit 14 **TDE**: *Trigger DMA request enable.*

0: Trigger DMA request disabled.
1: Trigger DMA request enabled.

Bit 13 Reserved, always read as 0

Bit 12 **CC4DE**: *Capture/Compare 4 DMA request enable.*

0: CC4 DMA request disabled.
1: CC4 DMA request enabled.

Bit 11 **CC3DE**: *Capture/Compare 3 DMA request enable.*

0: CC3 DMA request disabled.
1: CC3 DMA request enabled.

Bit 10 **CC2DE**: *Capture/Compare 2 DMA request enable.*

0: CC2 DMA request disabled.
1: CC2 DMA request enabled.

Bit 9 **CC1DE**: *Capture/Compare 1 DMA request enable.*

0: CC1 DMA request disabled.
1: CC1 DMA request enabled.

Bit 8 **UDE**: *Update DMA request enable.*

0: Update DMA request disabled.
1: Update DMA request enabled.

Bit 7 Reserved, always read as 0.

Bit 6 **TIE**: *Trigger interrupt enable.*

0: Trigger interrupt disabled.
1: Trigger interrupt enabled.

Bit 5 Reserved, always read as 0.

Bit 4 **CC4IE**: *Capture/Compare 4 interrupt enable.*

0: CC4 interrupt disabled.
1: CC4 interrupt enabled.

Bit 3 **CC3IE**: *Capture/Compare 3 interrupt enable.*

0: CC3 interrupt disabled.
1: CC3 interrupt enabled.

Bit 2 **CC2IE**: *Capture/Compare 2 interrupt enable.*

0: CC2 interrupt disabled.
1: CC2 interrupt enabled.

Bit 1 **CC1IE**: *Capture/Compare 1 interrupt enable.*

0: CC1 interrupt disabled.
1: CC1 interrupt enabled.

Bit 0 **UIE**: *Update interrupt enable.*

0: Update interrupt disabled.
1: Update interrupt enabled.

13.4.5 Status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4 OF	CC3 OF	CC2 OF	CC1 OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 15:13 Reserved, always read as 0.

Bit 12 **CC4OF**: *Capture/Compare 4 Overcapture Flag.*

refer to CC1OF description

Bit 11 **CC3OF**: *Capture/Compare 3 Overcapture Flag.*

refer to CC1OF description

Bit 10 **CC2OF**: *Capture/Compare 2 Overcapture Flag.*

refer to CC1OF description

Bit 9 **CC1OF**: *Capture/Compare 1 Overcapture Flag.*

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to ‘0’.
0: No overcapture has been detected.
1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, always read as 0.

Bit 6 **TIF**: *Trigger interrupt Flag.*

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode, both edges in case gated mode is selected). It is cleared by software.
0: No trigger event occurred.
1: Trigger interrupt pending.

Bit 5 Reserved, always read as 0

Bit 4 **CC4IF**: *Capture/Compare 4 interrupt Flag.*

refer to CC1IF description

Bit 3 **CC3IF**: *Capture/Compare 3 interrupt Flag.*

refer to CC1IF description

Bit 2 **CC2IF**: *Capture/Compare 2 interrupt Flag.*

refer to CC1IF description

Bit 1 **CC1IF**: *Capture/compare 1 interrupt Flag.*

If channel CC1 is configured as output:
This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description). It is cleared by software.
0: No match.
1: The content of the counter TIMx_CNT has matched the content of the TIMx_CCR1 register.

If channel CC1 is configured as input:
This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.
0: No input capture occurred.
1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity).

Bit 0 **UIF**: *Update interrupt flag.*

This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value (update if REP_CNT=0) and if the UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

13.4.6 Event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
									w		w	w	w	w	w

Bits 15:7 Reserved, always read as 0.

Bit 6 **TG**: *Trigger generation.*

This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action.
1: The TIF flag is set in TIMx_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 Reserved, always read as 0.

Bit 4 **CC4G**: *Capture/compare 4 generation.*

refer to CC1G description

Bit 3 **CC3G**: *Capture/compare 3 generation.*

refer to CC1G description

Bit 2 **CC2G**: *Capture/compare 2 generation.*

refer to CC1G description

Bit 1 **CC1G**: *Capture/compare 1 generation.*

This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action.
1: A capture/compare event is generated on channel 1:
If channel CC1 is configured as output:
CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.
If channel CC1 is configured as input:
The current value of the counter is captured in TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: *Update generation.*

This bit can be set by software, it is automatically cleared by hardware.
0: No action.
1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

13.4.7 Capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]		OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Output compare mode

Bit 15 **OC2CE:** *Output Compare 2 Clear Enable*

Bits 14:12 **OC2M[2:0]**: *Output Compare 2 Mode.*

Bit 11 **OC2PE**: *Output Compare 2 Preload enable.*

Bit 10 **OC2FE**: *Output Compare 2 Fast enable.*

Bits 9:8 **CC2S[1:0]**: *Capture/Compare 2 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC2 channel is configured as output.
01: CC2 channel is configured as input, IC2 is mapped on TI2.
10: CC2 channel is configured as input, IC2 is mapped on TI1.
11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = ’0’ in TIMx_CCER).

Bit 7 **OC1CE:** *Output Compare 1Clear Enable*

OC1CE: Output Compare 1 Clear Enable
0: OC1Ref is not affected by the ETRF input
1: OC1Ref is cleared as soon as a High level is detected on ETRF input

Bits 6:4 **OC1M**: *Output Compare 1 Mode.*

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.
000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.
001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.
100: Force inactive level - OC1REF is forced low.
101: Force active level - OC1REF is forced high.
110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF=‘0’) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=’1’).
111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

Note 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S=’00’ (the channel is configured in output).

Note 2: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.

Bit 3 **OC1PE**: *Output Compare 1 Preload enable.*

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.
1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

Note 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S='00' (the channel is configured in output).

Note 2: The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.

Bit 2 **OC1FE**: *Output Compare 1 Fast enable.*

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.
0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.
1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: *Capture/Compare 1 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC1 channel is configured as output.
01: CC1 channel is configured as input, IC1 is mapped on TI1.
10: CC1 channel is configured as input, IC1 is mapped on TI2.
11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).

Input capture mode

Bits 15:12 **IC2F**: *Input Capture 2 Filter.*

Bits 11:10 **IC2PSC[1:0]**: *Input Capture 2 Prescaler.*

Bits 9:8 **CC2S**: *Capture/Compare 2 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC2 channel is configured as output.
01: CC2 channel is configured as input, IC2 is mapped on TI2.
10: CC2 channel is configured as input, IC2 is mapped on TI1.
11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCER).

Bits 7:4 **IC1F**: *Input Capture 1 Filter.*

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS}.
0001: $f_{SAMPLING}=f_{CK_INT}$, N=2.
0010: $f_{SAMPLING}=f_{CK_INT}$, N=4.
0011: $f_{SAMPLING}=f_{CK_INT}$, N=8.
0100: $f_{SAMPLING}=f_{DTS}/2$, N=6.
0101: $f_{SAMPLING}=f_{DTS}/2$, N=8.
0110: $f_{SAMPLING}=f_{DTS}/4$, N=6.
0111: $f_{SAMPLING}=f_{DTS}/4$, N=8.
1000: $f_{SAMPLING}=f_{DTS}/8$, N=6.
1001: $f_{SAMPLING}=f_{DTS}/8$, N=8.
1010: $f_{SAMPLING}=f_{DTS}/16$, N=5.
1011: $f_{SAMPLING}=f_{DTS}/16$, N=6.
1100: $f_{SAMPLING}=f_{DTS}/16$, N=8.
1101: $f_{SAMPLING}=f_{DTS}/32$, N=5.
1110: $f_{SAMPLING}=f_{DTS}/32$, N=6.
1111: $f_{SAMPLING}=f_{DTS}/32$, N=8.

Note: In current silicon revision, f_{DTS} is replaced in the formula by CK_INT when ICxF[3:0]= 1, 2 or 3.

Bits 3:2 **IC1PSC**: *Input Capture 1 Prescaler.*

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).
The prescaler is reset as soon as CC1E='0' (TIMx_CCER register).
00: no prescaler, capture is done each time an edge is detected on the capture input.
01: capture is done once every 2 events.
10: capture is done once every 4 events.
11: capture is done once every 8 events.

Bits 1:0 **CC1S**: *Capture/Compare 1 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC1 channel is configured as output.
01: CC1 channel is configured as input, IC1 is mapped on TI1.
10: CC1 channel is configured as input, IC1 is mapped on TI2.
11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).

13.4.8 Capture/compare mode register 2 (TIMx_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4 CE	OC4M[2:0]			OC4 PE	OC4 FE	CC4S[1:0]		OC3 CE.	OC3M[2:0]			OC3 PE	OC3 FE	CC3S[1:0]	
IC4F[3:0]				IC4PSC[1:0]				IC3F[3:0]				IC3PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Output Compare mode

Bit 15 **OC4CE:** Output Compare 4 Clear Enable

Bits 14:12 **OC4M**: *Output Compare 4 Mode.*

Bit 11 **OC4PE**: *Output Compare 4 Preload enable.*

Bit 10 **OC4FE**: *Output Compare 4 Fast enable.*

Bits 9:8 **CC4S**: *Capture/Compare 4 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC4 channel is configured as output.
01: CC4 channel is configured as input, IC4 is mapped on TI4.
10: CC4 channel is configured as input, IC4 is mapped on TI3.
11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = ’0’ in TIMx_CCER).

Bit 7 **OC3CE:** Output Compare 3 Clear Enable

Bits 6:4 **OC3M**: *Output Compare 3 Mode.*

Bit 3 **OC3PE**: *Output Compare 3 Preload enable.*

Bit 2 **OC3FE**: *Output Compare 3 Fast enable.*

Bits 1:0 **CC3S**: *Capture/Compare 3 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC3 channel is configured as output.
01: CC3 channel is configured as input, IC3 is mapped on TI3.
10: CC3 channel is configured as input, IC3 is mapped on TI4.
11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = ’0’ in TIMx_CCER).

Input capture mode

Bits 15:12 **IC4F**: *Input Capture 4 Filter.*

Bits 11:10 **IC4PSC**: *Input Capture 4 Prescaler.*

Bits 9:8 **CC4S**: *Capture/Compare 4 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC4 channel is configured as output.
01: CC4 channel is configured as input, IC4 is mapped on TI4.
10: CC4 channel is configured as input, IC4 is mapped on TI3.
11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx_CCER).

Bits 7:4 **IC3F**: *Input Capture 3 Filter.*

Bits 3:2 **IC3PSC**: *Input Capture 3 Prescaler.*

Bits 1:0 **CC3S**: *Capture/Compare 3 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC3 channel is configured as output.
01: CC3 channel is configured as input, IC3 is mapped on TI3.
10: CC3 channel is configured as input, IC3 is mapped on TI4.
11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx_CCER).

13.4.9 Capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4P	CC4E	Reserved		CC3P	CC3E	Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
		rw	rw			rw	rw			rw	rw			rw	rw

Bits 15:14 Reserved, always read as 0.

Bit 13 **CC4P**: *Capture/Compare 4 output Polarity.*

refer to CC1P description

Bit 12 **CC4E**: *Capture/Compare 4 output enable.*

refer to CC1E description

Bits 11:10 Reserved, always read as 0.

Bit 9 **CC3P**: *Capture/Compare 3 output Polarity.*

refer to CC1P description

Bit 8 **CC3E**: *Capture/Compare 3 output enable.*

refer to CC1E description

Bits 7:6 Reserved, always read as 0.

Bit 5 **CC2P**: *Capture/Compare 2 output Polarity.*

refer to CC1P description

Bit 4 **CC2E**: *Capture/Compare 2 output enable.*

refer to CC1E description

Bits 3:2 Reserved, always read as 0.

Bit 1 **CC1P**: *Capture/Compare 1 output Polarity.*

CC1 channel configured as output:
0: OC1 active high.
1: OC1 active low.
CC1 channel configured as input:
This bit selects whether IC1 or IC1 is used for trigger or capture operations.
0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted.
1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*

CC1 channel configured as output:
0: Off - OC1 is not active.
1: On - OC1 signal is output on the corresponding output pin.

CC1 channel configured as input:
This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.
0: Capture disabled.
1: Capture enabled.

Table 60. Output control bit for standard OCx channels

CCxE bit	OCx output state
0	Output Disabled (OCx=0, OCx_EN=0)
1	OCx=OCxREF + Polarity, OCx_EN=1

Note: The state of the external I/O pins connected to the standard OCx channels depends on the OCx channel state and the GPIO and AFIO registers.

13.4.10 Counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CNT[15:0]**: *Counter Value.*

13.4.11 Prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: *Prescaler Value.*

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).

PSC contains the value to be loaded in the active prescaler register at each update event.

13.4.12 Auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: *Prescaler Value.*

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 13.3.1: Time-base unit on page 268* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

13.4.13 Capture/compare register 1 (TIMx_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: *Capture/Compare 1 Value).*

If channel CC1 is configured as output:

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC1 output.

If channel CC1is configured as input:

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

13.4.14 Capture/compare register 2 (TIMx_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: *Capture/Compare 2 Value.*

If channel CC2 is configured as output:
CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC2 output.

If channel CC2 is configured as input:
CCR2 is the counter value transferred by the last input capture 2 event (IC2).

13.4.15 Capture/compare register 3 (TIMx_CCR3)

Address offset: 0x3C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR3[15:0]**: *Capture/Compare Value.*

If channel CC3 is configured as output:
CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR3 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC3 output.

If channel CC3is configured as input:
CCR3 is the counter value transferred by the last input capture 3 event (IC3).

13.4.16 Capture/compare register 4 (TIMx_CCR4)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]**: *Capture/Compare Value.*

1/ if CC4 channel is configured as output (CC4S bits):
CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR4 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC4 output.

2/ if CC4 channel is configured as input (CC4S bits in TIMx_CCMR4 register):
CCR4 is the counter value transferred by the last input capture 4 event (IC4).

13.4.17 DMA control register (TIMx_DCR)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			DBL[4:0]					Reserved			DBA[4:0]				
			rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

Bits 15:13 Reserved, always read as 0

Bits 12:8 **DBL[4:0]**: *DMA Burst Length.*

This 5-bits vector defines the length of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address), i.e. the number of bytes to be transferred.
00000: 1 byte,
00001: 2 bytes,
00010: 3 bytes,
...
10001: 18 bytes.

Bits 7:5 Reserved, always read as 0

Bits 4:0 **DBA[4:0]**: *DMA Base Address.*

This 5-bit vector defines the base-address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register.
Example:
00000: TIMx_CR1,
00001: TIMx_CR2,
00010: TIMx_SMCR,
...

13.4.18 DMA address for full transfer (TIMx_DMAR)

Address offset: 0x4C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAB[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **DMAB[15:0]**: *DMA register for burst accesses.*

A read or write access to the DMAR register accesses the register located at the address:

"(TIMx_CR1 address) + DBA + (DMA index)" in which:

TIMx_CR1 address is the address of the control register 1,

DBA is the DMA base address configured in the TIMx_DCR register,

DMA index is the offset automatically controlled by the DMA transfer, depending on the length of the transfer DBL in the TIMx_DCR register.

13.4.19 TIMx register map

TIMx registers are mapped as 16-bit addressable registers as described in the table below:

Table 61. TIMx register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**TIMx_CR1**	Reserved																						CKD [1:0]		ARPE	CMS [1:0]		DIR	OPM	URS	UDIS	CEN
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x04	**TIMx_CR2**	Reserved																								TI1S	MMS[2:0]			CCDS	Reserved		
	Reset value																									0	0	0	0	0			
0x08	**TIMx_SMCR**	Reserved																ETP	ECE	ETPS [1:0]		ETF[3:0]				MSM	TS[2:0]			Reserved	SMS[2:0]		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
0x0C	**TIMx_DIER**	Reserved																	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Reserved	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	Reset value																		0	0	0	0	0	0	0		0	0	0	0	0	0	0
0x10	**TIMx_SR**	Reserved																			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
	Reset value																				0	0	0	0			0	0	0	0	0	0	0
0x14	**TIMx_EGR**	Reserved																									TG	Reserved	CC4G	CC3G	CC2G	CC1G	UG
	Reset value																										0		0	0	0	0	0
0x18	**TIMx_CCMR1** *Output Compare mode*	Reserved																OC2CE	OC2M [2:0]			OC2PE	OC2FE	CC2S [1:0]		OC1CE	OC1M [2:0]			OC1PE	OC1FE	CC1S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TIMx_CCMR1 *Input Capture mode*	Reserved																IC2F[3:0]				IC2 PSC [1:0]		CC2S [1:0]		IC1F[3:0]				IC1 PSC [1:0]		CC1S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 61. TIMx register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	**TIMx_CCMR2** *Output Compare mode*	Reserved																O24CE	OC4M [2:0]			OC4PE	OC4FE	CC4S [1:0]		OC3CE	OC3M [2:0]			OC3PE	OC3FE	CC3S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TIMx_CCMR2 *Input Capture mode*	Reserved																IC4F[3:0]				IC4 PSC [1:0]		CC4S [1:0]		IC3F[3:0]				IC3 PSC [1:0]		CC3S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	**TIMx_CCER**	Reserved																		CC4P	CC4E	Reserved		CC3P	CC3E	Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
	Reset value																			0	0			0	0			0	0			0	0
0x24	**TIMx_CNT**	Reserved																CNT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	**TIMx_PSC**	Reserved																PSC[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**TIMx_ARR**	Reserved																ARR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x30	Reserved																																
0x34	**TIMx_CCR1**	Reserved																CCR1[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x38	**TIMx_CCR2**	Reserved																CCR2[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	**TIMx_CCR3**	Reserved																CCR3[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x40	**TIMx_CCR4**	Reserved																CCR4[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44	Reserved																																
0x48	**TIMx_DCR**	Reserved																				DBL[4:0]				Reserved				DBA[4:0]			
	Reset value																					0	0	0	0					0	0	0	0
0x4C	**TIMx_DMAR**	Reserved																DMAB[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

14 Basic timer (TIM6&7)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to high-density STM32F101xx and STM32F103xx devices only.

14.1 TIM6&7 introduction

The basic timers TIM6 and TIM7 consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used as generic timers for time-base generation but they are also specifically used to drive the digital-to-analog converter (DAC). In fact, the timers are internally connected to the DAC and are able to drive it through their trigger outputs.

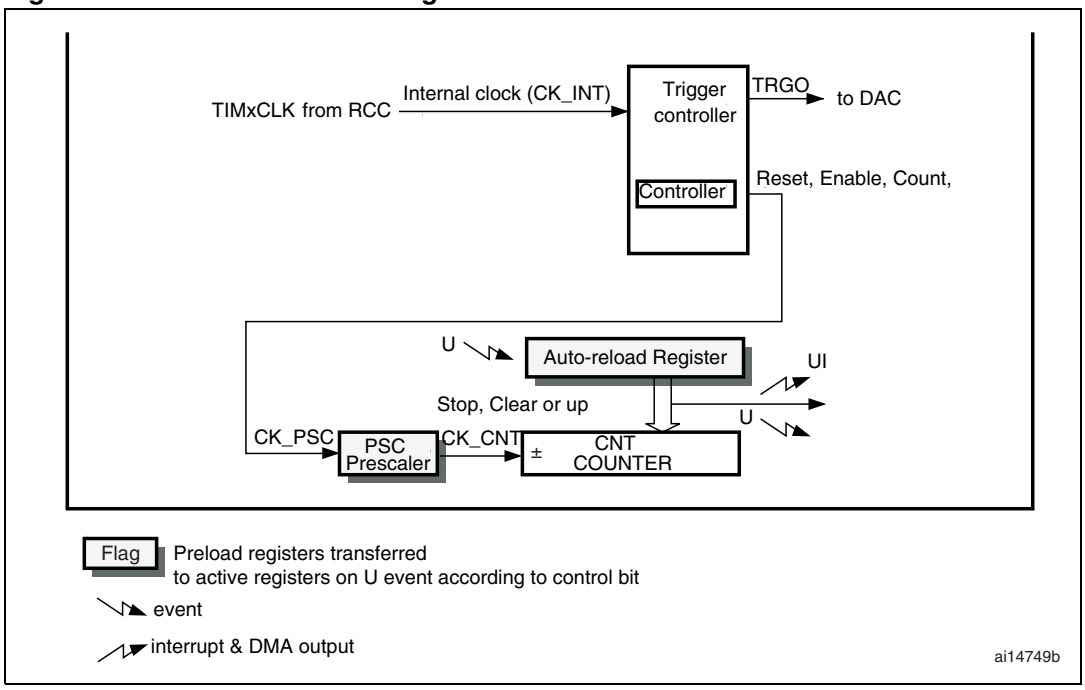
The timers are completely independent, and do not share any resources.

14.2 TIM6&TIM7 main features

Basic timer (TIM6&7) features include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535
- Synchronization circuit to trigger the DAC
- Interrupt/DMA generation on the update event: counter overflow

Figure 140. Basic timer block diagram



14.3 TIM6&TIM7 functional description

14.3.1 Time-base unit

The main block of the programmable timer is a 16-bit upcounter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC)
- Auto-Reload Register (TIMx_ARR)

The auto-reload register is preloaded. The preload register is accessed each time an attempt is made to write or read the auto-reload register. The contents of the preload register are transferred into the shadow register permanently or at each update event UEV, depending on the auto-reload preload enable bit (ARPE) in the TIMx_CR1 register. The update event is sent when the counter reaches the overflow value and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in the TIMx_CR1 register is set.

Note that the actual counter enable signal CNT_EN is set 1 clock cycle after CEN.

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It can be changed on the fly as the TIMx_PSC control register is buffered. The new prescaler ratio is taken into account at the next update event.

Figure 141 and *Figure 142* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

Figure 141. Counter timing diagram with prescaler division change from 1 to 2

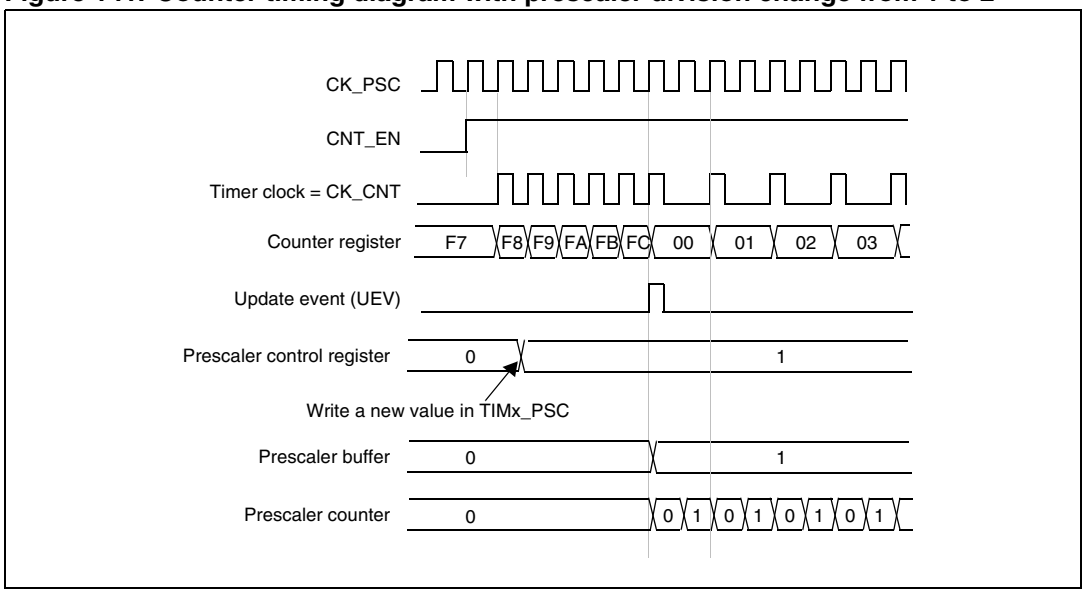
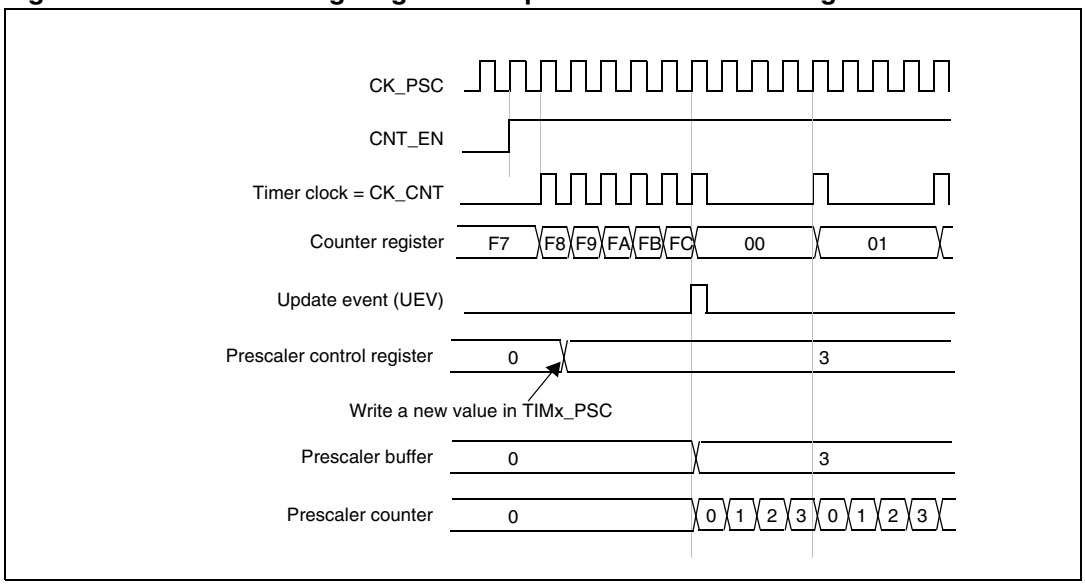


Figure 142. Counter timing diagram with prescaler division change from 1 to 4



14.3.2 Counting mode

The counter counts from 0 to the auto-reload value (contents of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

An update event can be generate at each counter overflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller).

The UEV event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This avoids updating the shadow registers while writing new values into the preload registers. In this way, no update event occurs until the UDIS bit has been written to 0, however, the counter and the prescaler counter both restart from 0 (but the prescale rate does not change). In addition, if the URS (update request selection) bit in the TIMx_CR1 register is set, setting the UG bit generates an update event UEV, but the UIF flag is not set (so no interrupt or DMA request is sent).

When an update event occurs, all the registers are updated and the update flag (UIF bit in the TIMx_SR register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (contents of the TIMx_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR = 0x36.

Figure 143. Counter timing diagram, internal clock divided by 1

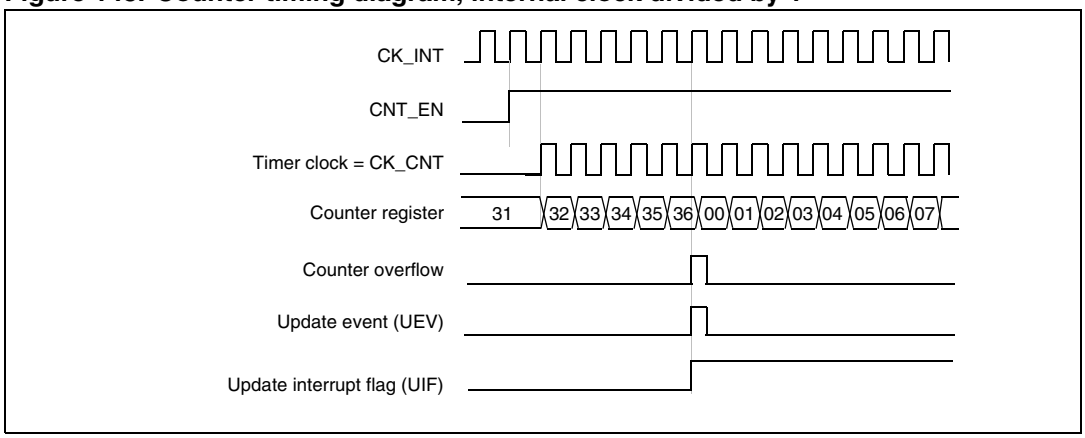


Figure 144. Counter timing diagram, internal clock divided by 2

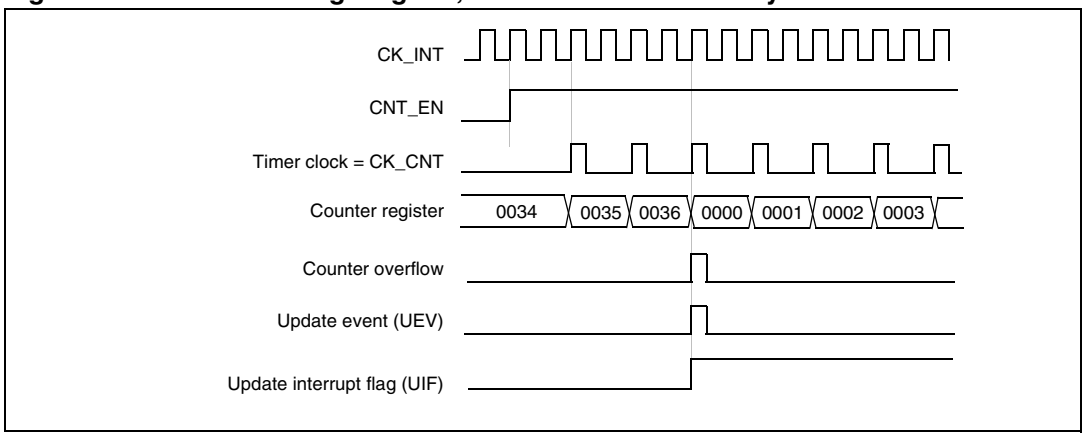


Figure 145. Counter timing diagram, internal clock divided by 4

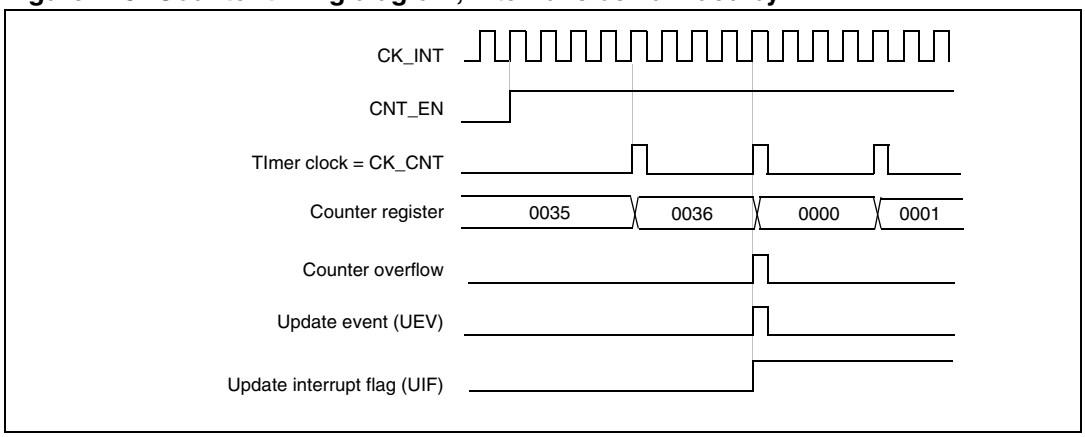


Figure 146. Counter timing diagram, internal clock divided by N

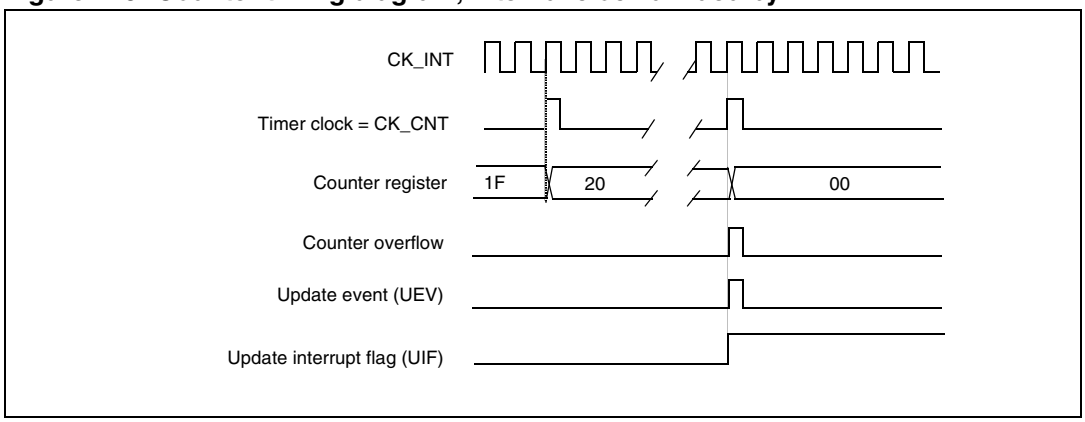


Figure 147. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded)

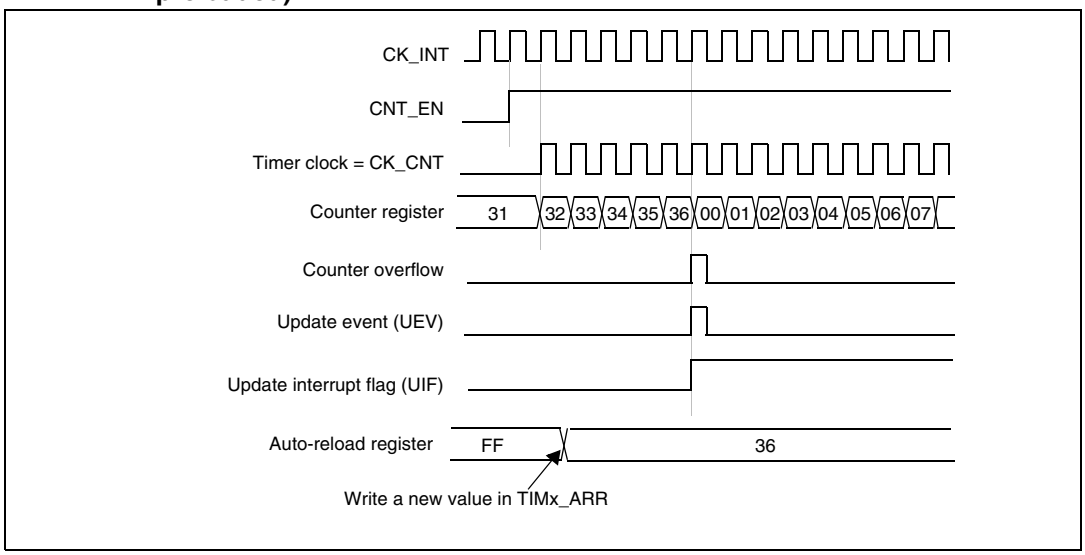
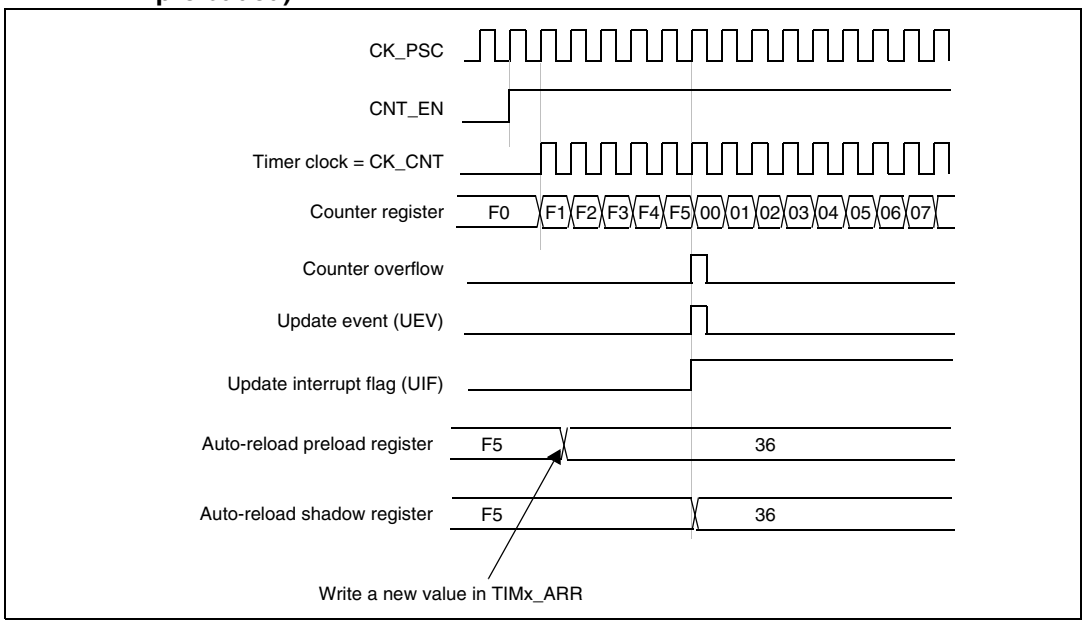


Figure 148. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)



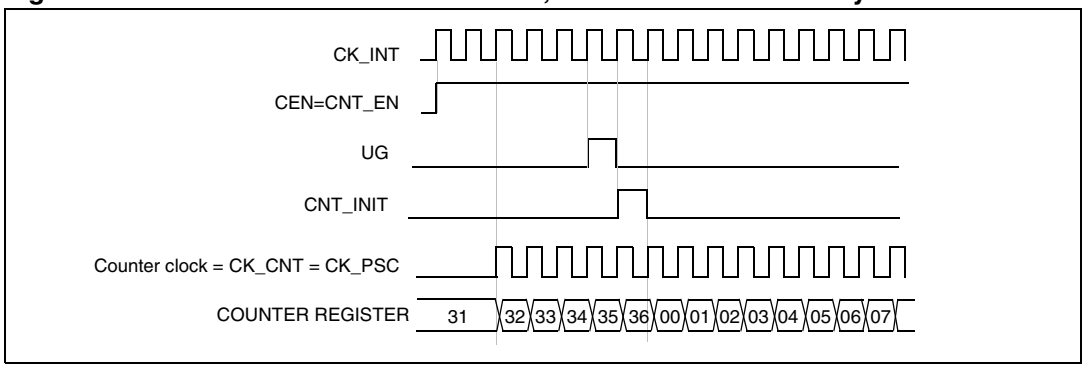
14.3.3 Clock source

The counter clock is provided by the Internal clock (CK_INT) source.

The CEN (in the TIMx_CR1 register) and UG bits (in the TIMx_EGR register) are actual control bits and can be changed only by software (except for UG that remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK_INT.

Figure 149 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 149. Control circuit in normal mode, internal clock divided by 1



14.3.4 Debug mode

When the microcontroller enters the debug mode (Cortex-M3 core - halted), the TIMx counter either continues to work normally or stops, depending on the DBG_TIMx_STOP configuration bit in the DBG module. For more details, refer to *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C*.

14.4 TIM6&TIM7 registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

14.4.1 Control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								ARPE	Reserved			OPM	URS	UDIS	CEN
								rw				rw	rw	rw	rw

Bits 15:8 Reserved, always read as 0

Bit 7 **ARPE**: *Auto-Reload Preload enable.*

0: TIMx_ARR register is not buffered.
1: TIMx_ARR register is buffered.

Bits 6:4 Reserved, always read as 0

Bit 3 **OPM**: *One-Pulse Mode.*

0: Counter is not stopped at update event
1: Counter stops counting at the next update event (clearing the CEN bit).

Bit 2 **URS**: *Update Request Source.*

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generates an update interrupt or DMA request if enabled. These events can be:

– Counter overflow/underflow
– Setting the UG bit
– Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: *Update Disable.*

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

– Counter overflow/underflow
– Setting the UG bit
– Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: *Counter enable.*

0: Counter disabled
1: Counter enabled

Note: Gated mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

14.4.2 Control register 2 (TIMx_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									MMS[2:0]			Reserved			
									rw	rw	rw				

Bits 15:7 Reserved, always read as 0.

Bits 6:4 **MMS**: *Master Mode Selection.*

These bits are used to select the information to be sent in master mode to slave timers for synchronization (TRGO). The combination is as follows:
000: **Reset** - the UG bit from the TIMx_EGR register is used as a trigger output (TRGO). If reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT_EN, is used as a trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in the TIMx_SMCR register).

010: **Update** - The update event is selected as a trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

Bits 3:0 Reserved, always read as 0

14.4.3 DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							UDE	Reserved							UIE
							rw								rw

Bit 15:9 Reserved, always read as 0.

Bit 8 **UDE**: *Update DMA request enable.*

0: Update DMA request disabled.
1: Update DMA request enabled.

Bit 7:1 Reserved, always read as 0.

Bit 0 **UIE**: *Update interrupt enable.*

0: Update interrupt disabled.
1: Update interrupt enabled.

14.4.4 Status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															UIF
															rc_w0

Bits 15:1 Reserved, always read as 0.

Bit 0 **UIF**: *Update interrupt flag.*

This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value and if UDIS = 0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in the TIMx_EGR register, if URS = 0 and UDIS = 0 in the TIMx_CR1 register.

14.4.5 Event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															UG
															w

Bits 15:1 Reserved, always read as 0.

Bit 0 **UG**: *Update generation.*

This bit can be set by software, it is automatically cleared by hardware.
0: No action.
1: Re-initializes the timer counter and generates an update of the registers. Note that the prescaler counter is cleared too (but the prescaler ratio is not affected).

14.4.6 Counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CNT[15:0]**: *Counter Value.*

14.4.7 Prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: *Prescaler Value.*
The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).

PSC contains the value to be loaded into the active prescaler register at each update event.

14.4.8 Auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: *Prescaler Value.*
ARR is the value to be loaded into the actual auto-reload register.
Refer to *Section 14.3.1: Time-base unit on page 323* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

14.4.9 TIM6&7 register map

TIMx registers are mapped as 16-bit addressable registers as described in the table below:

Table 62. TIM6&7 - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**TIMx_CR1**	Reserved																								ARPE	Reserved			OPM	URS	UDIS	CEN
	Reset value																									0				0	0	0	0
0x04	**TIMx_CR2**	Reserved																									MMS[2:0]			Reserved			
	Reset value																										0	0	0				
0x08	Reserved																																
0x0C	**TIMx_DIER**	Reserved																							UDE	Reserved							UIE
	Reset value																								0								0
0x10	**TIMx_SR**	Reserved																															UIF
	Reset value																																0
0x14	**TIMx_EGR**	Reserved																															UG
	Reset value																																0
0x18	Reserved																																
0x1C	Reserved																																
0x20	Reserved																																
0x24	**TIMx_CNT**	Reserved																CNT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	**TIMx_PSC**	Reserved																PSC[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**TIMx_ARR**	Reserved																ARR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

15 Real-time clock (RTC)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

15.1 RTC introduction

The real-time clock is an independent timer. The RTC provides a set of continuously running counters which can be used, with suitable software, to provide a clock-calendar function. The counter values can be written to set the current time/date of the system.

The RTC core and clock configuration (RCC_BDCR register) are in the Backup domain, which means that RTC setting and time are kept after reset or wakeup from Standby mode.

After reset, access to the Backup registers and RTC is disabled and the Backup domain (BKP) is protected against possible parasitic write access. To enable access to the Backup registers and the RTC, proceed as follows:

- enable the power and backup interface clocks by setting the PWREN and BKPEN bits in the RCC_APB1ENR register
- set the DBP bit the Power Control Register (PWR_CR) to enable access to the Backup registers and RTC.

15.2 RTC main features

- Programmable prescaler: division factor up to 2^{20}
- 32-bit programmable counter for long-term measurement
- Two separate clocks: PCLK1 for the APB1 interface and RTC clock (must be at least four times slower than the PCLK1 clock)
- The RTC clock source could be any of the following three:
  - HSE clock divided by 128
  - LSE oscillator clock
  - LSI oscillator clock (refer to *Section 6.2.8: RTC clock* for details)
- Two separate reset types:
  - The APB1 interface is reset by system reset
  - The RTC Core (Prescaler, Alarm, Counter and Divider) is reset only by a Backup domain reset (see *Section 6.1.3: Backup domain reset on page 67*).
- Three dedicated maskable interrupt lines:
  - Alarm interrupt, for generating a software programmable alarm interrupt.
  - Seconds interrupt, for generating a periodic interrupt signal with a programmable period length (up to 1 second).
  - Overflow interrupt, to detect when the internal programmable counter rolls over to zero.

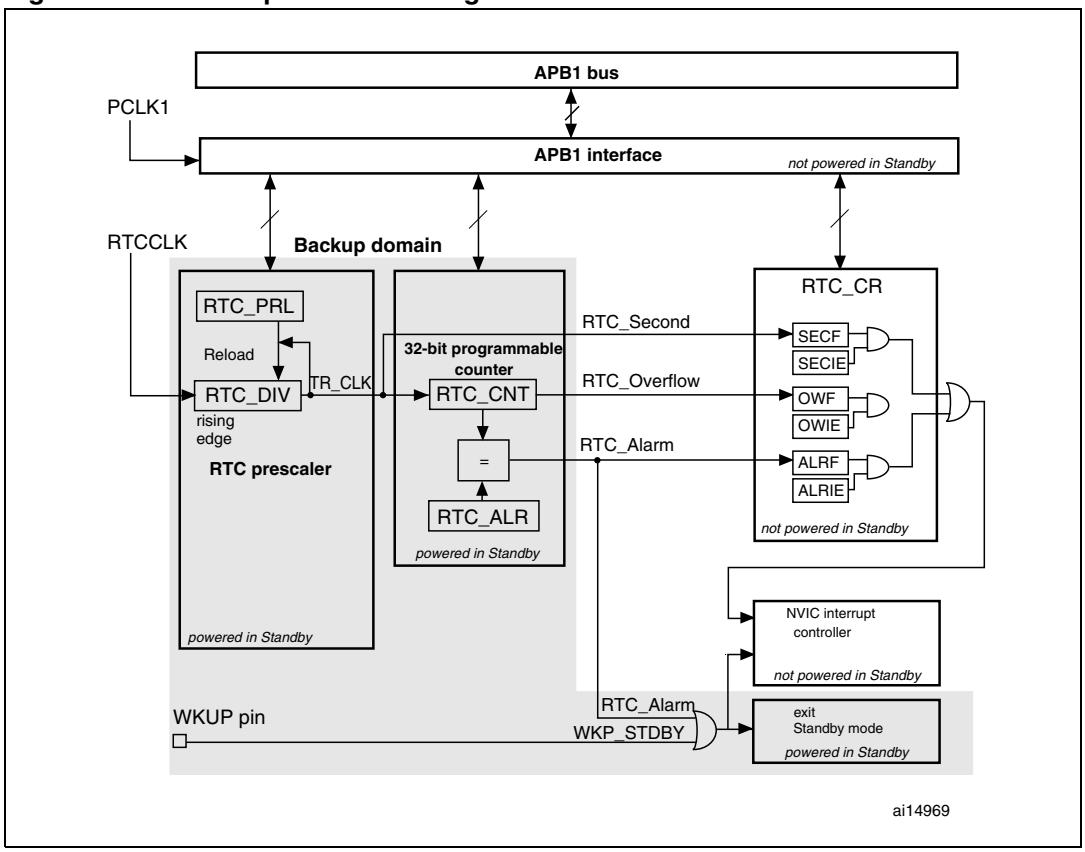
15.3 RTC functional description

15.3.1 Overview

The RTC consists of two main units (see *Figure 150 on page 336*). The first one (APB1 Interface) is used to interface with the APB1 bus. This unit also contains a set of 16-bit registers accessible from the APB1 bus in read or write mode (for more information refer to *Section 15.4: RTC registers on page 339*). The APB1 interface is clocked by the APB1 bus clock in order to interface with the APB1 bus.

The other unit (RTC Core) consists of a chain of programmable counters made of two main blocks. The first block is the RTC prescaler block, which generates the RTC time base TR_CLK that can be programmed to have a period of up to 1 second. It includes a 20-bit programmable divider (RTC Prescaler). Every TR_CLK period, the RTC generates an interrupt (Second Interrupt) if it is enabled in the RTC_CR register. The second block is a 32-bit programmable counter that can be initialized to the current system time. The system time is incremented at the TR_CLK rate and compared with a programmable date (stored in the RTC_ALR register) in order to generate an alarm interrupt, if enabled in the RTC_CR control register.

Figure 150. RTC simplified block diagram



15.3.2 Resetting RTC registers

All system registers are asynchronously reset by a System Reset or Power Reset, except for RTC_PRL, RTC_ALR, RTC_CNT, and RTC_DIV.

The RTC_PRL, RTC_ALR, RTC_CNT, and RTC_DIV registers are reset only by a Backup Domain reset. Refer to *Section 6.1.3 on page 67*.

15.3.3 Reading RTC registers

The RTC core is completely independent from the RTC APB1 interface.

Software accesses the RTC prescaler, counter and alarm values through the APB1 interface but the associated readable registers are internally updated at each rising edge of the RTC clock resynchronized by the RTC APB1 clock. This is also true for the RTC flags.

This means that the first read to the RTC APB1 registers may be corrupted (generally read as 0) if the APB1 interface has previously been disabled and the read occurs immediately after the APB1 interface is enabled but before the first internal update of the registers. This can occur if:

- A system reset or power reset has occurred
- The MCU has just woken up from Standby mode (see *Section 4.3: Low-power modes*)
- The MCU has just woken up from Stop mode (see *Section 4.3: Low-power modes*)

In all the above cases, the RTC core has been kept running while the APB1 interface was disabled (reset, not clocked or unpowered).

Consequently when reading the RTC registers, after having disabled the RTC APB1 interface, the software must first wait for the RSF bit (Register Synchronized Flag) in the RTC_CRL register to be set by hardware.

Note that the RTC APB1 interface is not affected by WFI and WFE low-power modes.

15.3.4 Configuring RTC registers

To write in the RTC_PRL, RTC_CNT, RTC_ALR registers, the peripheral must enter Configuration Mode. This is done by setting the CNF bit in the RTC_CRL register.

In addition, writing to any RTC register is only enabled if the previous write operation is finished. To enable the software to detect this situation, the RTOFF status bit is provided in the RTC_CR register to indicate that an update of the registers is in progress. A new value can be written to the RTC registers only when the RTOFF status bit value is ’1’.

Configuration procedure:

1. Poll RTOFF, wait until its value goes to ‘1’
2. Set the CNF bit to enter configuration mode
3. Write to one or more RTC registers
4. Clear the CNF bit to exit configuration mode
5. Poll RTOFF, wait until its value goes to ‘1’ to check the end of the write operation.

The write operation only executes when the CNF bit is cleared; it takes at least three RTCCLK cycles to complete.

15.3.5 RTC flag assertion

The RTC Second flag (SECF) is asserted on each RTC Core clock cycle before the update of the RTC Counter.

The RTC Overflow flag (OWF) is asserted on the last RTC Core clock cycle before the counter reaches 0x0000.

The RTC_Alarm and RTC Alarm flag (ALRF) (see *Figure 151*) are asserted on the last RTC Core clock cycle before the counter reaches the RTC Alarm value stored in the Alarm register increased by one (RTC_ALR + 1). The write operation in the RTC Alarm and RTC Second flag must be synchronized by using one of the following sequences:

- Use the RTC Alarm interrupt and inside the RTC interrupt routine, the RTC Alarm and/or RTC Counter registers are updated.
- Wait for SECF bit to be set in the RTC Control register. Update the RTC Alarm and/or the RTC Counter register.

Figure 151. RTC second and alarm waveform example with PR=0003, ALARM=00004

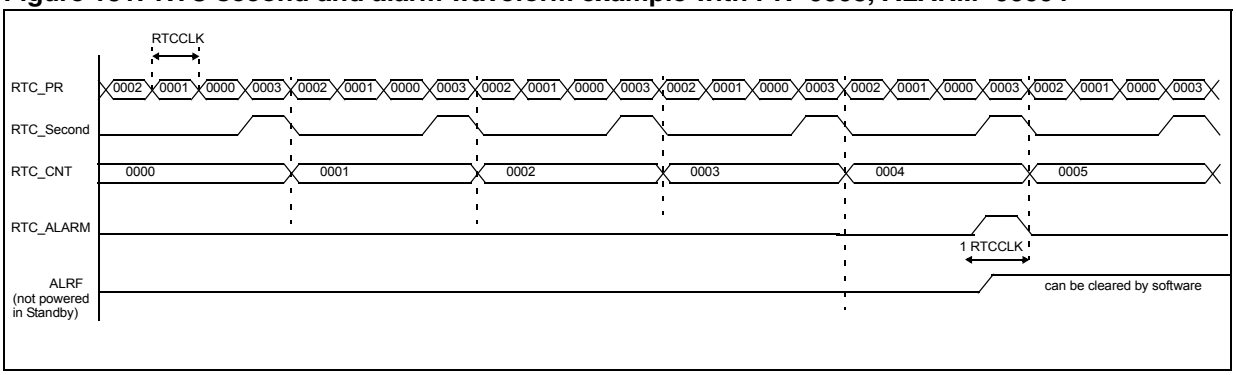
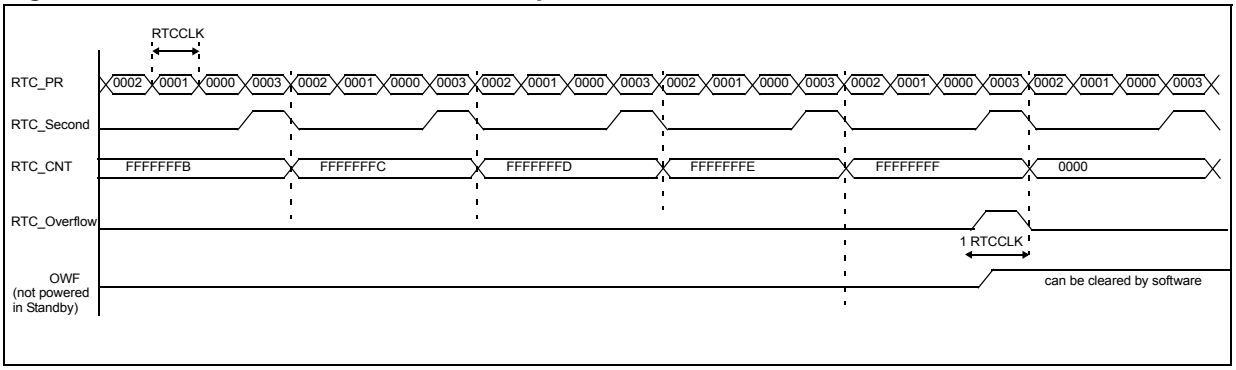


Figure 152. RTC Overflow waveform example with PR=0003



15.4 RTC registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

15.4.1 RTC control register high (RTC_CRH)

Address offset: 0x00
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													OWIE	ALRIE	SECIE
													rw	rw	rw

Bits 15:3 Reserved, forced by hardware to 0.

Bit 2 **OWIE:** *OverfloW Interrupt Enable*
0: Overflow interrupt is masked.
1: Overflow interrupt is enabled.

Bit 1 **ALRIE**: *Alarm Interrupt Enable*
0: Alarm interrupt is masked.
1: Alarm interrupt is enabled.

Bit 0 **SECIE:** *Second Interrupt Enable*
0: Second interrupt is masked.
1: Second interrupt is enabled.

These bits are used to mask interrupt requests. Note that at reset all interrupts are disabled, so it is possible to write to the RTC registers to ensure that no interrupt requests are pending after initialization. It is not possible to write to the RTC_CRH register when the peripheral is completing a previous write operation (flagged by RTOFF=0, see *Section 15.3.4 on page 337*).

The RTC functions are controlled by this control register. Some bits must be written using a specific configuration procedure (see *Configuration procedure:*).

15.4.2 RTC control register low (RTC_CRL)

Address offset: 0x04
Reset value: 0x0020

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RTOFF	CNF	RSF	OWF	ALRF	SECF
										r	rw	rc_w0	rc_w0	rc_w0	rc_w0

Bits 15:6 Reserved, forced by hardware to 0.

Bit 5 **RTOFF:** *RTC operation OFF*

With this bit the RTC reports the status of the last write operation performed on its registers, indicating if it has been completed or not. If its value is ‘0’ then it is not possible to write to any of the RTC registers. This bit is read only.
0: Last write operation on RTC registers is still ongoing.
1: Last write operation on RTC registers terminated.

Bit 4 **CNF**: *Configuration Flag*

This bit must be set by software to enter in configuration mode so as to allow new values to be written in the RTC_CNT, RTC_ALR or RTC_PRL registers. The write operation is only executed when the CNF bit is reset by software after has been set.
0: Exit configuration mode (start update of RTC registers).
1: Enter configuration mode.

Bit 3 **RSF:** *Registers Synchronized Flag*

This bit is set by hardware at each time the RTC_CNT and RTC_DIV registers are updated and cleared by software. Before any read operation after an APB1 reset or an APB1 clock stop, this bit must be cleared by software, and the user application must wait until it is set to be sure that the RTC_CNT, RTC_ALR or RTC_PRL registers are synchronized.
0: Registers not yet synchronized.
1: Registers synchronized.

Bit 2 **OWF:** *OverfloW Flag*

This bit is set by hardware when the 32-bit programmable counter overflows. An interrupt is generated if OWIE=1 in the RTC_CRH register. It can be cleared only by software. Writing ‘1’ has no effect.
0: Overflow not detected
1: 32-bit programmable counter overflow occurred.

Bit 1 **ALRF:** *Alarm Flag*

This bit is set by hardware when the 32-bit programmable counter reaches the threshold set in the RTC_ALR register. An interrupt is generated if ALRIE=1 in the RTC_CRH register. It can be cleared only by software. Writing ‘1’ has no effect.
0: Alarm not detected
1: Alarm detected

Bit 0 **SECF:** *Second Flag*

This bit is set by hardware when the 32-bit programmable prescaler overflows, thus incrementing the RTC counter. Hence this flag provides a periodic signal with a period corresponding to the resolution programmed for the RTC counter (usually one second). An interrupt is generated if SECIE=1 in the RTC_CRH register. It can be cleared only by software. Writing ‘1’ has no effect.
0: Second flag condition not met.
1: Second flag condition met.

The functions of the RTC are controlled by this control register. It is not possible to write to the RTC_CR register while the peripheral is completing a previous write operation (flagged by RTOFF=0, see *Section 15.3.4 on page 337*).

Note: 1 *Any flag remains pending until the appropriate RTC_CR request bit is reset by software, indicating that the interrupt request has been granted.*

2 *At reset the interrupts are disabled, no interrupt requests are pending and it is possible to write to the RTC registers.*

3 *The OWF, ALRF, SECF and RSF bits are not updated when the APB1 clock is not running.*

4 *The OWF, ALRF, SECF and RSF bits can only be set by hardware and only cleared by software.*

5 *If ALRF = 1 and ALRIE = 1, the RTC global interrupt is enabled. If EXTI Line 17 is also enabled through the EXTI Controller, both the RTC global interrupt and the RTC Alarm interrupt are enabled.*

6 *If ALRF = 1, the RTC Alarm interrupt is enabled if EXTI Line 17 is enabled through the EXTI Controller in interrupt mode. When the EXTI Line 17 is enabled in event mode, a pulse is generated on this line (no RTC Alarm interrupt generation).*

15.4.3 RTC prescaler load register (RTC_PRLH / RTC_PRLL)

The Prescaler Load registers keep the period counting value of the RTC prescaler. They are write-protected by the RTOFF bit in the RTC_CR register, and a write operation is allowed if the RTOFF value is ‘1’.

RTC prescaler load register high (RTC_PRLH)

Address offset: 0x08
Write only (see *Section 15.3.4 on page 337*)
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												PRL[19:16]			
												w	w	w	w

Bits 15:4 Reserved, forced by hardware to 0.

Bits 3:0 **PRL[19:16]:** *RTC Prescaler Reload Value High*
These bits are used to define the counter clock frequency according to the following formula:
$f_{TR_CLK} = f_{RTCCLK}/(PRL[19:0]+1)$
Caution: The zero value is not recommended. RTC interrupts and flags cannot be asserted correctly.

RTC prescaler load register low (RTC_PRLL)

Address offset: 0x0C
Write only (see *Section 15.3.4 on page 337*)
Reset value: 0x8000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRL[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 15:0 **PRL[15:0]:** *RTC Prescaler Reload Value Low*
These bits are used to define the counter clock frequency according to the following formula:
$f_{TR_CLK} = f_{RTCCLK}/(PRL[19:0]+1)$

Note: If the input clock frequency (f_{RTCCLK}) is 32.768 kHz, write 7FFFh in this register to get a signal period of 1 second.

15.4.4 RTC prescaler divider register (RTC_DIVH / RTC_DIVL)

During each period of TR_CLK, the counter inside the RTC prescaler is reloaded with the value stored in the RTC_PRL register. To get an accurate time measurement it is possible to read the current value of the prescaler counter, stored in the RTC_DIV register, without stopping it. This register is read-only and it is reloaded by hardware after any change in the RTC_PRL or RTC_CNT registers.

RTC prescaler divider register high (RTC_DIVH)

Address offset: 0x10
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												RTC_DIV[19:16]			
												r	r	r	r

Bits 15:4 Reserved

Bits 3:0 **RTC_DIV[19:16]:** *RTC Clock Divider High*

RTC prescaler divider register low (RTC_DIVL)

Address offset: 0x14
Reset value: 0x8000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_DIV[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **RTC_DIV[15:0]:** *RTC Clock Divider Low*

15.4.5 RTC counter register (RTC_CNTH / RTC_CNTL)

The RTC core has one 32-bit programmable counter, accessed through two 16-bit registers; the count rate is based on the TR_CLK time reference, generated by the prescaler. RTC_CNT registers keep the counting value of this counter. They are write-protected by bit RTOFF in the RTC_CR register, and a write operation is allowed if the RTOFF value is '1'. A write operation on the upper (RTC_CNTH) or lower (RTC_CNTL) registers directly loads the corresponding programmable counter and reloads the RTC Prescaler. When reading, the current value in the counter (system date) is returned.

RTC counter register high (RTC_CNTH)

Address offset: 0x18
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_CNT[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **RTC_CNT[31:16]:** *RTC Counter High*

Reading the RTC_CNTH register, the current value of the high part of the RTC Counter register is returned. To write to this register it is necessary to enter configuration mode (see *Section 15.3.4: Configuring RTC registers on page 337*).

RTC counter register low (RTC_CNTL)

Address offset: 0x1C
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **RTC_CNT[15:0]:** *RTC Counter Low*

Reading the RTC_CNTL register, the current value of the lower part of the RTC Counter register is returned. To write to this register it is necessary to enter configuration mode (see *Section 15.3.4: Configuring RTC registers on page 337*).

15.4.6 RTC alarm register high (RTC_ALRH / RTC_ALRL)

When the programmable counter reaches the 32-bit value stored in the RTC_ALR register, an alarm is triggered and the RTC_alarmIT interrupt request is generated. This register is write-protected by the RTOFF bit in the RTC_CR register, and a write operation is allowed if the RTOFF value is '1'.

RTC alarm register high (RTC_ALRH)

Address offset: 0x20
Write only (see *Section 15.3.4 on page 337*)
Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_ALR[31:16]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 15:0 **RTC_ALR[31:16]:** *RTC Alarm High*

The high part of the alarm time is written by software in this register. To write to this register it is necessary to enter configuration mode (see *Section 15.3.4: Configuring RTC registers on page 337*).

RTC alarm register low (RTC_ALRL)

Address offset: 0x24
Write only (see *Section 15.3.4 on page 337*)
Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_ALR[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 15:0 **RTC_ALR[15:0]:** *RTC Alarm Low*

The low part of the alarm time is written by software in this register. To write to this register it is necessary to enter configuration mode (see *Section 15.3.4: Configuring RTC registers on page 337*).

15.4.7 RTC register map

RTC registers are mapped as 16-bit addressable registers as described in the table below:

Table 63. RTC - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	**RTC_CRH**	Reserved																													OWIE	ALRIE	SECIE
	Reset value																														0	0	0
0x004	**RTC_CRL**	Reserved																										RTOFF	CNF	RSF	OWF	ALRF	SECF
	Reset value																											1	0	0	0	0	0
0x008	**RTC_PRLH**	Reserved																												PRL[19:16]			
	Reset value																													0	0	0	0
0x00C	**RTC_PRLL**	Reserved																PRL[15:0]															
	Reset value																	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x010	**RTC_DIVH**	Reserved																DIV[31:16]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x014	**RTC_DIVL**	Reserved																DIV[15:0]															
	Reset value																	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x018	**RTC_CNTH**	Reserved																CNT[13:16]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x01C	**RTC_CNTL**	Reserved																CNT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x020	**RTC_ALRH**	Reserved																ALR[31:16]															
	Reset value																	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x024	**RTC_ALRL**	Reserved																ALR[15:0]															
	Reset value																	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Refer to *Table 1 on page 35* for the register boundary addresses.

16 Independent watchdog (IWDG)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

16.1 IWDG introduction

The STM32F10xxx has two embedded watchdog peripherals which offer a combination of high safety level, timing accuracy and flexibility of use. Both watchdog peripherals (Independent and Window) serve to detect and resolve malfunctions due to software failure, and to trigger system reset or an interrupt (window watchdog only) when the counter reaches a given timeout value.

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (40 kHz) and thus stays active even if the main clock fails. The window watchdog (WWDG) clock is prescaled from the APB1 clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

The IWDG is best suited to applications which require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. The WWDG is best suited to applications which require the watchdog to react within an accurate timing window. For further information on the window watchdog, refer to *Section 17 on page 352*.

16.2 IWDG main features

- Free-running downcounter
- clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Reset (if watchdog activated) when the downcounter value of 0x000 is reached

16.3 IWDG functional description

Figure 153 shows the functional blocks of the independent watchdog module.

When the independent watchdog is started by writing the value 0xCCCC in the Key register (IWDG_KR), the counter starts counting down from the reset value of 0xFFF. When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).

Whenever the key value 0xAAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.

16.3.1 Hardware watchdog

If the “Hardware watchdog” feature is enabled through the device option bits, the watchdog is automatically enabled at power-on, and will generate a reset unless the Key register is written by the software before the counter reaches end of count.

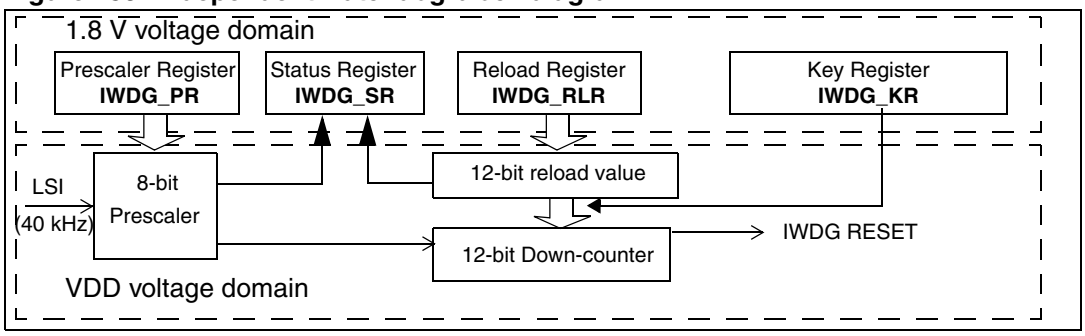
16.3.2 Register access protection

Write access to the IWDG_PR and IWDG_RLR registers is protected. To modify them, you must first write the code 0x5555 in the IWDG_KR register. A write access to this register with a different value will break the sequence and register access will be protected again. This implies that it is the case of the reload operation (writing 0xAAAA).
A status register is available to indicate that an update of the prescaler or the down-counter reload value is on going.

16.3.3 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module. For more details, refer to *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C*.

Figure 153. Independent watchdog block diagram



Note: The watchdog function is implemented in the V_{DD} voltage domain that is still functional in Stop and Standby modes.

Table 64. Watchdog timeout period (with 40 kHz input clock)(1)

Prescaler divider	PR[2:0] bits	Min timeout (ms) RL[11:0]= 0x000	Max timeout (ms) RL[11:0]= 0xFFF
/4	0	0.1	409.6
/8	1	0.2	819.2
/16	2	0.4	1638.4
/32	3	0.8	3276.8
/64	4	1.6	6553.6
/128	5	3.2	13107.2
/256	6 (or 7)	6.4	26214.4

1. These timings are given for a 40 kHz clock but the microcontroller’s internal RC frequency can vary from 30 to 60 kHz. Moreover, given an exact RC oscillator frequency, the exact timings still depend on the phasing of the APB interface clock versus the RC oscillator 40 kHz clock so that there is always a full RC period of uncertainty.

The LSI can be calibrated so as to compute the IWDG timeout with an acceptable accuracy. For more details refer to *LSI calibration on page 71*.

16.4 IWDG registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

16.4.1 Key register (IWDG_KR)

Address offset: 0x00

Reset value: 0x0000 0000 (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved, read as 0.

Bits 15:0 **KEY[15:0]:** *Key value (write only, read 0000h)*

These bits must be written by software at regular intervals with the key value AAAAh, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 5555h to enables access to the IWDG_PR and IWDG_RLR registers (see *Section 16.3.2*)

Writing the key value CCCCh starts the watchdog (except if the hardware watchdog option is selected)

16.4.2 Prescaler register (IWDG_PR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													PR[2:0]		
													rw	rw	rw

Bits 31:3 Reserved, read as 0.

Bits 2:0 **PR[2:0]:** *Prescaler divider*

These bits are write access protected see*Section 16.3.2*. They are written by software to select the prescaler divider feeding the counter clock. PVU bit of IWDG_SR must be reset in order to be able to change the prescaler divider.

000: divider /4
001: divider /8
010: divider /16
011: divider /32
100: divider /64
101: divider /128
110: divider /256
111: divider /256

Note: Reading this register returns the prescaler value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the PVU bit in the IWDG_SR register is reset.

16.4.3 Reload register (IWDG_RLR)

Address offset: 0x08

Reset value: 0x0000 0FFF (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				RL[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, read as 0.

Bits11:0 **RL[11:0]:** Watchdog counter reload value

These bits are write access protected see *Section 16.3.2*. They are written by software to define the value to be loaded in the watchdog counter each time the value AAAAh is written in the IWDG_KR register. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to *Table 64.*

The RVU bit in the IWDG_SR register must be reset in order to be able to change the reload value.

Note: Reading this register returns the reload value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on this register. For this reason the value read from this register is valid only when the RVU bit in the IWDG_SR register is reset.

16.4.4 Status register (IWDG_SR)

Address offset: 0x0C

Reset value: 0x0000 0000 (not reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														RVU	PVU
														r	r

Bits 31:2 Reserved

Bit 1 **RVU:** Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V_{DD} voltage domain (takes up to 5 RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU:** Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V_{DD} voltage domain (takes up to 5 RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.

Note: If several reload values or prescaler values are used by application, it is mandatory to wait until RVU bit is reset before changing the reload value and to wait until PVU bit is reset before changing the prescaler value. However, after updating the prescaler and/or the reload value it is not necessary to wait until RVU or PVU is reset before continuing code execution (even in case of low-power mode entry, the write operation is taken into account and will complete)

16.4.5 IWDG register map

The following table gives the IWDG register map and reset values.

Table 65. IWDG register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**IWDG_KR**	Reserved																KEY[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	**IWDG_PR**	Reserved																													PR[2:0]		
	Reset value																														0	0	0
0x08	**IWDG_RLR**	Reserved																				RL[11:0]											
	Reset value																					1	1	1	1	1	1	1	1	1	1	1	1
0x0C	**IWDG_SR**	Reserved																														RVU	PVU
	Reset value																															0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

17 Window watchdog (WWDG)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

17.1 WWDG introduction

The window watchdog is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence. The watchdog circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the contents of the downcounter before the T6 bit becomes cleared. An MCU reset is also generated if the 7-bit downcounter value (in the control register) is refreshed before the downcounter has reached the window register value. This implies that the counter must be refreshed in a limited window.

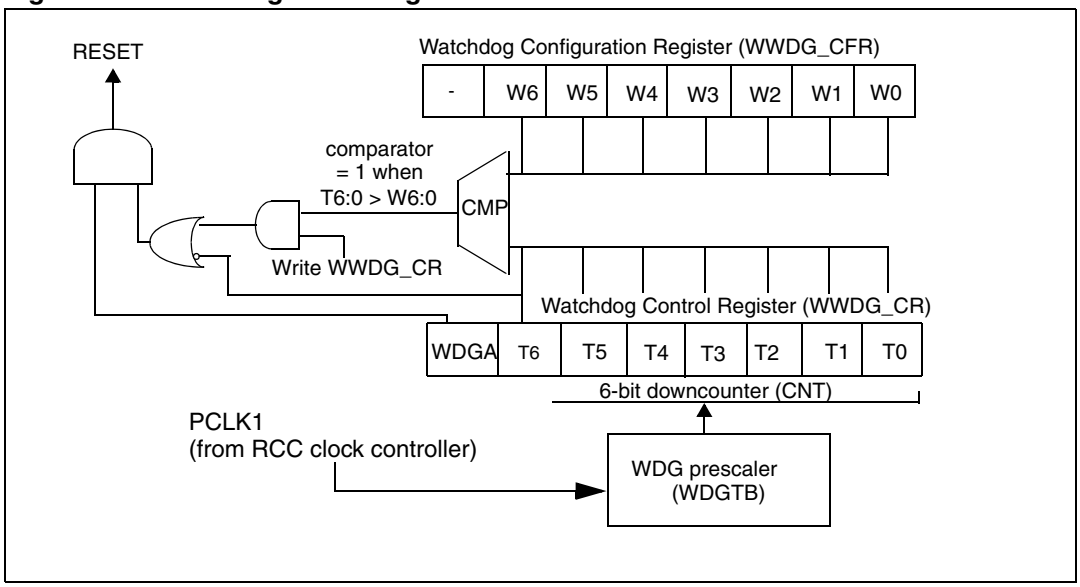
17.2 WWDG main features

- Programmable free-running downcounter
- Conditional reset
  - Reset (if watchdog activated) when the downcounter value becomes less than 40h
  - Reset (if watchdog activated) if the downcounter is reloaded outside the window (see *Figure 155*)
- Early wakeup interrupt (EWI): triggered (if enabled and the watchdog activated) when the downcounter is equal to 40h. Can be used to reload the counter and prevent WWDG reset

17.3 WWDG functional description

If the watchdog is activated (the WDGA bit is set in the WWDG_CR register) and when the 7-bit downcounter (T[6:0] bits) rolls over from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset. If the software reloads the counter while the counter is greater than the value stored in the window register, then a reset is generated.

Figure 154. Watchdog block diagram



The application program must write in the WWDG_CR register at regular intervals during normal operation to prevent an MCU reset. This operation must occur only when the counter value is lower than the window register value. The value to be stored in the WWDG_CR register must be between 0xFF and 0xC0:

- Enabling the watchdog:

  The watchdog is always disabled after a reset. It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset.

- Controlling the downcounter:

  This downcounter is free-running: It counts down even if the watchdog is disabled. When the watchdog is enabled, the T6 bit must be set to prevent generating an immediate reset.
  The T[5:0] bits contain the number of increments which represents the time delay before the watchdog produces a reset. The timing varies between a minimum and a maximum value due to the unknown status of the prescaler when writing to the WWDG_CR register (see *Figure 155*).
  The Configuration register (WWDG_CFR) contains the high limit of the window: To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x3F. *Figure 155* describes the window watchdog process.
  Another way to reload the counter is to use the early wakeup interrupt (EWI). This interrupt is enabled by setting the EWI bit in the WWDG_CFR register. When the downcounter reaches the value 40h, this interrupt is generated and the corresponding interrupt service routine (ISR) can be used to reload the counter to prevent WWDG reset.
  This interrupt is cleared by writing '0' to the EWIF bit in the WWDG_SR register.

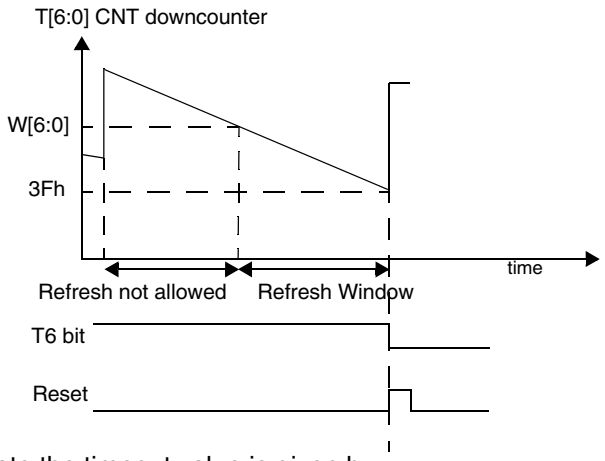
Note: The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).

17.4 How to program the watchdog timeout

Figure 155 shows the linear relationship between the 6-bit value to be loaded in the watchdog counter (CNT) and the resulting timeout duration in milliseconds. This can be used for a quick calculation without taking the timing variations into account. If more precision is needed, use the formulae in *Figure 155*.

Warning: When writing to the WWDG_CR register, always write 1 in the T6 bit to avoid generating an immediate reset.

Figure 155. Window watchdog timing diagram



The formula to calculate the timeout value is given by:

$$T_{WWDG} = T_{PCLK1} \times 4096 \times 2^{WDGTB} \times (T[5:0] + 1) \quad ;(ms)$$

where:

T_{WWDG}: WWDG timeout

T_{PCLK1}: APB1 Clock period measured in ms

Min-max timeout value @36MHz (PCLK1)

WDGTB	Min timeout value	Max timeout value
0	113 µs	7.28 ms
1	227 µs	14.56 ms
2	455 µs	29.12 ms
3	910 µs	58.25 ms

17.5 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core halted), the WWDG counter either continues to work normally or stops, depending on DBG_WWDG_STOP configuration bit in DBG module. For more details, refer to *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C*.

17.6 Debug registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

17.6.1 Control Register (WWDG_CR)

Address offset: 0x00

Reset value: 0x7F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								WDGA	T6	T5	T4	T3	T2	T1	T0
								rs	rw	rw	rw	rw	rw	rw	rw

Bits 31:8 Reserved

Bit 7 **WDGA:** *Activation bit*

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

0: Watchdog disabled

1: Watchdog enabled

Bits 6:0 **T[6:0]:** *7-bit counter (MSB to LSB).*

These bits contain the value of the watchdog counter. It is decremented every (4096 x 2^{WDGTB}) PCLK1 cycles. A reset is produced when it rolls over from 40h to 3Fh (T6 becomes cleared).

17.6.2 Configuration register (WWDG_CFR)

Address offset: 0x04

Reset value: 0x7F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						EWI	WDG TB1	WDG TB0	W6	W5	W4	W3	W2	W1	W0
						rs	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31:10 Reserved

Bit 9 **EWI:** *Early Wakeup Interrupt*

When set, an interrupt occurs whenever the counter reaches the value 40h. This interrupt is only cleared by hardware after a reset.

Bits 8:7 **WDGTB[1:0]:** *Timer Base*

The time base of the prescaler can be modified as follows:
00: CK Counter Clock (PCLK1 div 4096) div 1
01: CK Counter Clock (PCLK1 div 4096) div 2
10: CK Counter Clock (PCLK1 div 4096) div 4
11: CK Counter Clock (PCLK1 div 4096) div 8

Bits 6:0 **W[6:0]** *7-bit window value*

These bits contain the window value to be compared to the downcounter.

17.6.3 Status register (WWDG_SR)

Address offset: 0x08
Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															EWIF
															rc_w0

Bit 31:1Reserved

Bit 0 **EWIF:** *Early Wakeup Interrupt Flag*

This bit is set by hardware when the counter has reached the value 40h. 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.

17.6.4 WWDG register map

The following table gives the WWDG register map and reset values.

Table 66. WWDG register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**WWDG_CR**	Reserved																								WDGA	T[6:0]						
	Reset value																									0	1	1	1	1	1	1	1
0x04	**WWDG_CFR**	Reserved																						EWI	WDGTB1	WDGTB0	W[6:0]						
	Reset value																							0	0	0	1	1	1	1	1	1	1
0x08	**WWDG_SR**	Reserved																															EWIF
	Reset value																																0

Refer to *Table 1 on page 35* for the register boundary addresses.

18 Flexible static memory controller (FSMC)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 32 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to high-density devices only.

18.1 FSMC main features

The FSMC block is able to interface with synchronous and asynchronous memories and 16-bit PC memory cards. Its main purpose is to:

- translate the AHB transactions into the appropriate external device protocol
- meet the access timing requirements of the external devices

All external memories share the addresses, data and control signals with the controller. Each external device is accessed by means of a unique chip select. The FSMC performs only one access at a time to an external device.

The FSMC has the following main features:

- Interfaces with static memory-mapped devices including:
  - static random access memory (SRAM)
  - read-only memory (ROM)
  - NOR Flash memory
  - PSRAM (4 memory banks)
- Two banks of NAND Flash with ECC hardware that checks up to 8 Kbytes of data
- 16-bit PC Card compatible devices
- Supports burst mode access to synchronous devices (NOR Flash and PSRAM)
- 8- or 16-bit wide databus
- Independent chip select control for each memory bank
- Independent configuration for each memory bank
- Programmable timings to support a wide range of devices, in particular:
  - programmable wait states (up to 15)
  - programmable bus turnaround cycles (up to 15)
  - programmable output enable and write enable delays (up to 15)
  - independent read and write timings and protocol, so as to support the widest variety of memories and timings
- Write enable and byte lane select outputs for use with PSRAM and SRAM devices
- Translation of 32-bit wide AHB transactions into consecutive 16-bit or 8-bit accesses to external 16-bit or 8-bit devices

- Write FIFO, 16 words long, each word 32 bits wide. This makes it possible to write to slow memories and free the AHB quickly for other transactions. If a new transaction is started to the FSMC, first the FIFO is drained

The FSMC registers that define the external device type and associated characteristics are usually set at boot time and do not change until the next reset or power-up. However, it is possible to change the settings at any time.

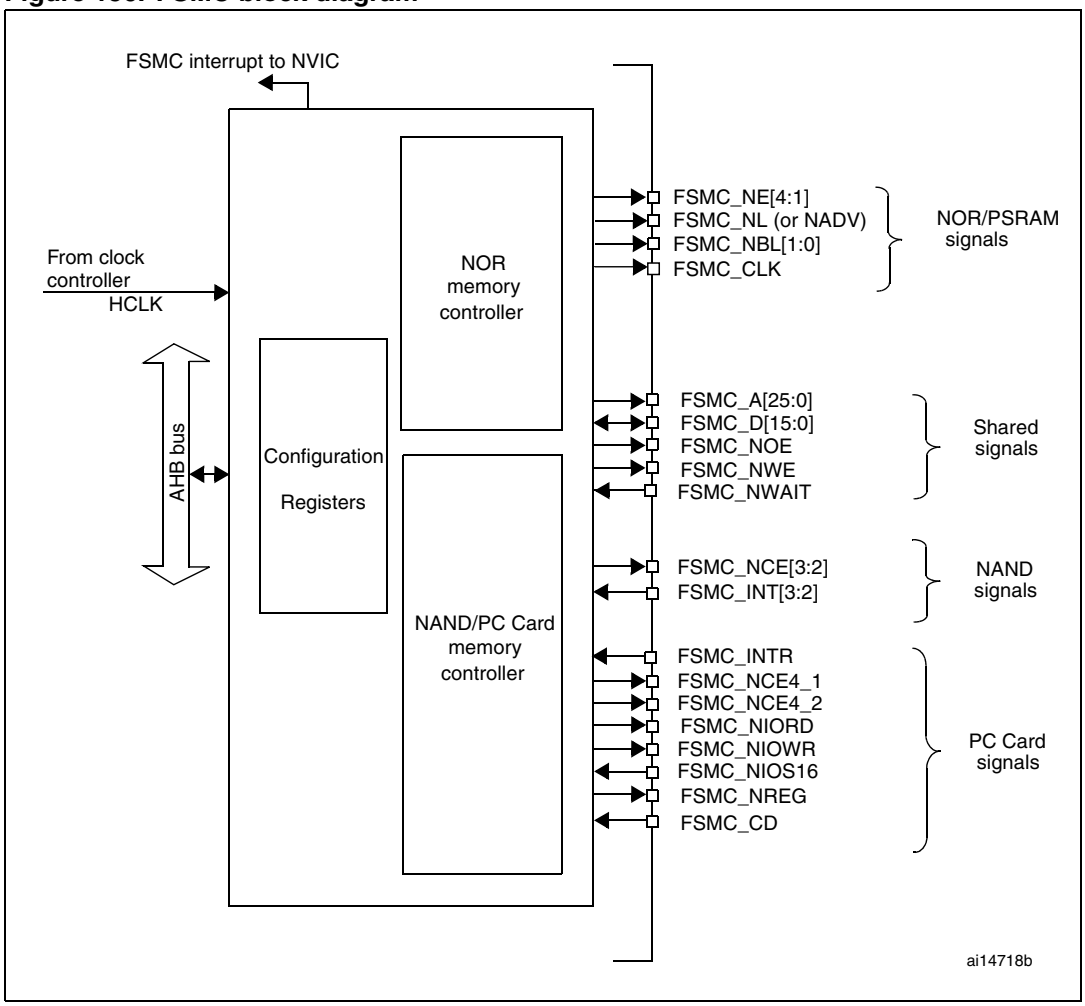
18.2 Block diagram

The FSMC consists of four main blocks:

- The AHB interface (including the FSMC configuration registers)
- The NOR Flash/PSRAM controller
- The NAND Flash/PC Card controller
- The external device interface

The block diagram is shown in *Figure 156*.

Figure 156. FSMC block diagram



18.3 AHB interface

The AHB slave interface enables internal CPUs and other bus master peripherals to access the external static memories.

AHB transactions are translated into the external device protocol. In particular, if the selected external memory is 16 or 8 bits wide, 32-bit wide transactions on the AHB are split into consecutive 16- or 8-bit accesses.

The AHB clock (HCLK) is the reference clock for the FSMC.

18.3.1 Supported memories and transactions

General transaction rules

The requested AHB transaction data size can be 8-, 16- or 32-bit wide whereas the accessed external device has a fixed data width. This may lead to inconsistent transfers.

Therefore, some simple transaction rules must be followed:

- AHB transaction size and memory data size are equal
  There is no issue in this case.
- AHB transaction size is greater than the memory size
  In this case, the FSMC splits the AHB transaction into smaller consecutive memory accesses in order to meet the external data width.
- AHB transaction size is smaller than the memory size
  Asynchronous transfers may or not be consistent depending on the type of external device.
  - Asynchronous accesses to devices that have the byte select feature (SRAM, ROM, PSRAM).
    In this case, the FSMC allows read/write transactions and accesses the right data through its byte lanes BL[1:0]
  - Asynchronous accesses to devices that do not have the byte select feature (NOR and NAND Flash 16-bit).
    This situation occurs when a byte access is requested to a 16-bit wide Flash memory. Clearly, the device cannot be accessed in byte mode (only 16-bit words can be read from/written to the Flash memory) therefore:

  a) Write transactions are not allowed

  b) Read transactions are allowed (the controller reads the entire 16-bit memory word and uses the needed byte only).

Configuration registers

The FSMC can be configured using a register set. See *Section 18.5.6*, for a detailed description of the NOR Flash/PSRAM controller registers. See *Section 18.6.7*, for a detailed description of the NAND Flash/PC Card registers.

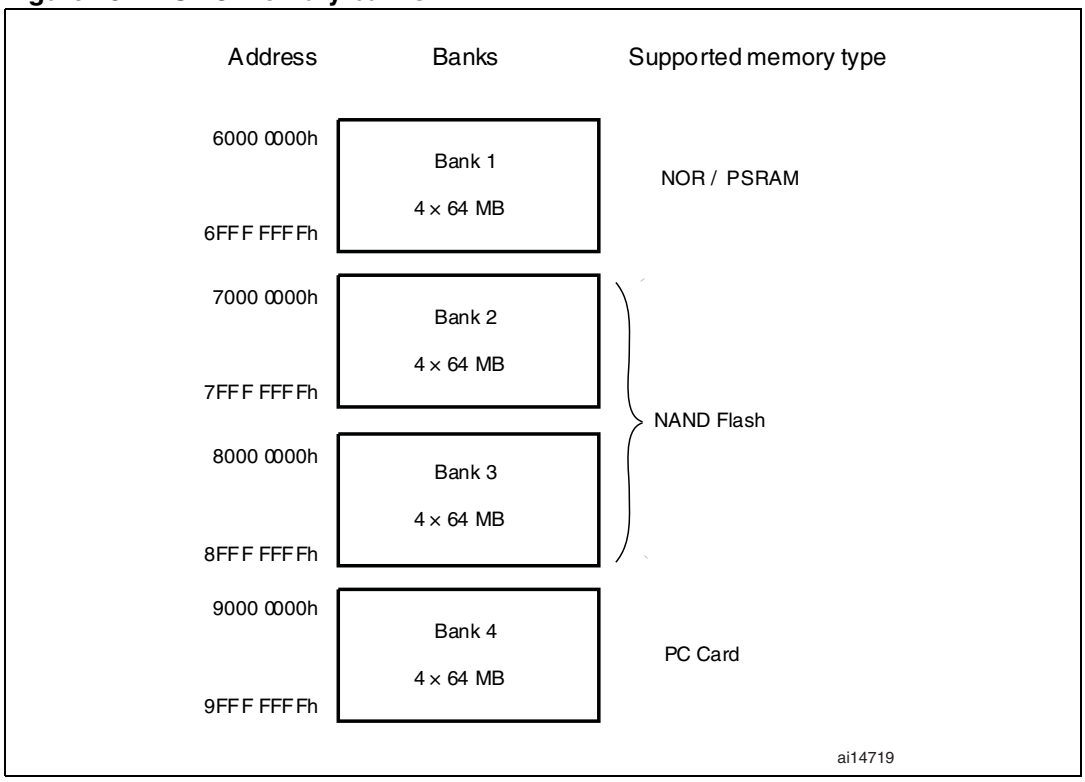
18.4 External device address mapping

From the FSMC point of view, the external memory is divided into 4 fixed-size banks of 256 Mbytes each (Refer to *Figure 157*):

- Bank 1 used to address up to 4 NOR Flash or PSRAM memory devices. This bank is split into 4 NOR/PSRAM regions with 4 dedicated Chip Select.
- Banks 2 and 3 used to address NAND Flash devices (1 device per bank)
- Bank 4 used to address a PC Card device

For each bank the type of memory to be used is user-defined in the Configuration register.

Figure 157. FSMC memory banks



18.4.1 NOR/PSRAM address mapping

HADDR[27:26] bits are used to select one of the four memory banks as shown in *Table 67*.

Table 67. NOR/PSRAM bank selection

HADDR[27:26][1]	Selected bank
00	Bank 1 NOR/PSRAM 1
01	Bank 1 NOR/PSRAM 2
10	Bank 1 NOR/PSRAM 3
11	Bank 1 NOR/PSRAM 4

1. HADDR are internal AHB address lines that are translated to external memory.

HADDR[25:0] contain the external memory address. Since HADDR is a byte address whereas the memory is addressed in words, the address actually issued to the memory varies according to the memory data width, as shown in the following table.

Table 68. External memory address

Memory width(1)	Data address issued to the memory	Maximum memory capacity (bits)
8-bit	HADDR[25:0]	64 Mbytes x 8 = 512 Mbit
16-bit	HADDR[25:1] >> 1	64 Mbytes/2 x 16 = 512 Mbit

1. In case of a 16-bit external memory width, the FSMC will internally use HADDR[25:1] to generate the address for external memory FSMC_A[24:0].
Whatever the external memory width (16-bit or 8-bit), FSMC_A[0] should be connected to external memory address A[0].

Wrap support for NOR Flash/PSRAM

Each NOR Flash/PSRAM memory bank can be configured to support wrap accesses.

On the memory side, two cases must be considered depending on the access mode: asynchronous or synchronous.

- **Asynchronous mode**: in this case, wrap accesses are fully supported as long as the address is supplied for every single access.
- **Synchronous mode**: in this case, the FSMC issues the address only once, and then the burst transfer is sequenced by the FSMC clock CLK.

  Some NOR memories support linear burst with wrap-around accesses, in which a fixed number of words is read from consecutive addresses modulo N (N is typically 8 or 16 and can be programmed through the NOR Flash configuration register). In this case, it is possible to set the memory wrap mode identical to the AHB master wrap mode.

Otherwise, in the case when the memory wrap mode and the AHB master wrap mode cannot be set identically, wrapping should be disabled (through the appropriate bit in the FSMC configuration register) and the wrap transaction split into two consecutive linear transactions.

18.4.2 NAND/PC Card address mapping

In this case, three banks are available, each of them divided into memory spaces as indicated in *Table 69*.

Table 69. Memory mapping and timing registers

Start address	End address	FSMC Bank	Memory space	Timing register
0x9C00 0000	0x9FFF FFFF	Bank 4 - PC card	I/O	FSMC_PIO4 (0xB0)
0x9800 0000	0x9BFF FFFF		Attribute	FSMC_PATT4 (0xAC)
0x9000 0000	0x93FF FFFF		Common	FSMC_PMEM4 (0xA8)
0x8800 0000	0x8BFF FFFF	Bank 3 - NAND Flash	Attribute	FSMC_PATT3 (0x8C)
0x8000 0000	0x83FF FFFF		Common	FSMC_PMEM3 (0x88)
0x7800 0000	0x7BFF FFFF	Bank 2- NAND Flash	Attribute	FSMC_PATT2 (0x6C)
0x7000 0000	0x73FF FFFF		Common	FSMC_PMEM2 (0x68)

For NAND Flash memory, the common and attribute memory spaces are subdivided into three sections (see in *Table 70* below) located in the lower 256 Kbytes:

- Data section (first 64 Kbytes in the common/attribute memory space)
- Command section (second 64 Kbytes in the common / attribute memory space)
- Address section (next 128 Kbytes in the common / attribute memory space)

Table 70. NAND bank selections

Section name	HADDR[17:16]	Address range
Address section	1X	0x020000-0x03FFFF
Command section	01	0x010000-0x01FFFF
Data section	00	0x000000-0x0FFFF

The application software uses the 3 sections to access the NAND Flash memory:

- **To send a command to NAND Flash memory**: the software must write the command value to any memory location in the command section.
- **To specify the NAND Flash address that must be read or written**: the software must write the address value to any memory location in the address section. Since an address can be 4 or 5 bytes long (depending on the actual memory size), several consecutive writes to the address section are needed to specify the full address.
- **To read or write data**: the software reads or writes the data value from or to any memory location in the data section.

Since the NAND Flash memory automatically increments addresses, there is no need to increment the address of the data section to access consecutive memory locations.

18.5 NOR Flash/PSRAM controller

The FSMC generates the appropriate signal timings to drive the following types of memories:

- Asynchronous SRAM and ROM
  - 8-bit
  - 16-bit
  - 32-bit
- PSRAM (Cellular RAM)
  - Asynchronous mode
  - Burst mode
- NOR Flash
  - Asynchronous mode or burst mode
  - Multiplexed or nonmultiplexed

The FSMC outputs a unique chip select signal NE[4:1] per bank. All the other signals (addresses, data and control) are shared.

For synchronous accesses, the FSMC issues the clock (CLK) to the selected external device. This clock is a submultiple of the HCLK clock. The size of each bank is fixed and equal to 64 Mbytes.

Each bank is configured by means of dedicated registers (see *Section 18.6.7*).

The programmable memory parameters include access timings (see *Table 71*) and support for wrap and wait management (for PSRAM and NOR Flash accessed in burst mode).

Table 71. Programmable NOR/PSRAM access parameters

Parameter	Function	Access mode	Unit	Min.	Max.
Address setup	Duration of the address setup phase	Asynchronous	AHB clock cycle (HCLK)	1	16
Address hold	Duration of the address hold phase	Asynchronous, muxed I/Os	AHB clock cycle (HCLK)	1	16
Data setup	Duration of the data setup phase	Asynchronous	AHB clock cycle (HCLK)	1	256
Bust turn	Duration of the bus turnaround phase	Asynchronous and synchronous read	AHB clock cycle (HCLK)	1	16
Clock divide ratio	Number of AHB clock cycles (HCLK) to build one memory clock cycle (CLK)	Synchronous	AHB clock cycle (HCLK)	1	16
Data latency	Number of clock cycles to issue to the memory before the first data of the burst	Synchronous	Memory clock cycle (CLK)	2	17

18.5.1 External memory interface signals

Table 72, *Table 73* and *Table 74* list the signals that are typically used to interface NOR Flash and PSRAM.

Note: Prefix "N". specifies the associated signal as active low.

NOR Flash, nonmultiplexed I/Os

Table 72. Nonmuxed I/O NOR Flash

FSMC signal name	I/O	Function
CLK	O	Clock (for synchronous burst)
A[25:0]	O	Address bus
D[15:0]	I/O	Bidirectional data bus
NE[x]	O	Chip select, x = 1..4
NOE	O	Output enable
NWE	O	Write enable
NWAIT	I	NOR Flash wait input signal to the FSMC

NOR Flash memories are addressed in 16-bit words. The maximum capacity is 512 Mbit (26 address lines).

NOR Flash, multiplexed I/Os

Table 73. Muxed I/O NOR Flash

FSMC signal name	I/O	Function
CLK	O	Clock (for synchronous burst)
A[25:16]	O	Address bus
AD[15:0]	I/O	16-bit multiplexed, bidirectional address/data bus
NE[x]	O	Chip select, x = 1..4
NOE	O	Output enable
NWE	O	Write enable
NL(=NADV)	O	Latch enable (this signal is called address valid, NADV, by some NOR Flash devices)
NWAIT	I	NOR Flash wait input signal to the FSMC

NOR-Flash memories are addressed in 16-bit words. The maximum capacity is 512 Mbit (26 address lines).

PSRAM

Table 74. PSRAM

FSMC signal name	I/O	Function
CLK	O	Clock (for synchronous burst)
A[25:0]	O	Address bus
D[15:0]	I/O	Data bidirectional bus
NE[x]	O	Chip select, x = 1..4 (called NCE by PSRAM (Cellular RAM i.e. CRAM))
NOE	O	Output enable
NWE	O	Write enable
NL(= NADV)	O	Address valid PSRAM input (memory signal name: NADV)
NWAIT	I	PSRAM wait input signal to the FSMC
NBL[1]	O	Upper byte enable (memory signal name: NUB)
NBL[0]	O	Lowed byte enable (memory signal name: NLB)

PSRAM memories are addressed in 16-bit words. The maximum capacity is 512 Mbit (26 address lines).

18.5.2 Supported memories and transactions

Table 75 below displays the supported devices, access modes and transactions. Transactions not allowed (or not supported) by the FSMC appear in gray.

Table 75. NOR Flash/PSRAM supported memories and transactions

Device	Mode	R/W	AHB data size	Memory data size	Allowed/ not allowed	Comments
NOR Flash (muxed I/Os and nonmuxed I/Os)	Asynchronous	R	8	16	Y	
	Asynchronous	W	8	16	N	
	Asynchronous	R	16	16	Y	
	Asynchronous	W	16	16	Y	
	Asynchronous	R	32	16	Y	Split into 2 FSMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FSMC accesses
	Asynchronous page	R	-	16	N	Mode is not supported
	Synchronous	R	8	16	N	
	Synchronous	R	16	16	Y	
	Synchronous	R	32	16	Y	
PSRAM (muxed I/Os and nonmuxed I/Os)	Asynchronous	R	8	16	Y	
	Asynchronous	W	8	16	Y	Use of byte lanes NBL[1:0]
	Asynchronous	R	16	16	Y	
	Asynchronous	W	16	16	Y	
	Asynchronous	R	32	16	Y	Split into 2 FSMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FSMC accesses
	Asynchronous page	R	-	16	N	Mode is not supported
	Synchronous	R	8	16	N	
	Synchronous	R	16	16	Y	
	Synchronous	R	32	16	Y	
SRAM and ROM	Asynchronous	R	8 / 16 / 32	8 / 16	Y	Use of byte lanes NBL[1:0]
	Asynchronous	W	8 / 16 / 32	8 / 16	Y	Use of byte lanes NBL[1:0]

18.5.3 General timing rules

Signals synchronization

- All controller output signals change on the rising edge of the internal clock (HCLK)
- In synchronous write mode (PSRAM devices), the output data changes on the falling edge of the memory clock (CLK)

18.5.4 NOR Flash/PSRAM controller timing diagrams

Asynchronous static memories (NOR Flash, SRAM)

- Signals are synchronized by the internal clock HCLK. This clock is not issued to the memory
- The FSMC always samples the data before de-asserting the chip select signal NE. This guarantees that the memory data-hold timing constraint is met (chip enable high to data transition, usually 0 ns min.)
- When extended mode is set, it is possible to mix modes A, B, C and D in read and write (it is for instance possible to read in mode A and write in mode B).

Mode 1 - SRAM/CRAM

Figure 158. Mode1 read accesses

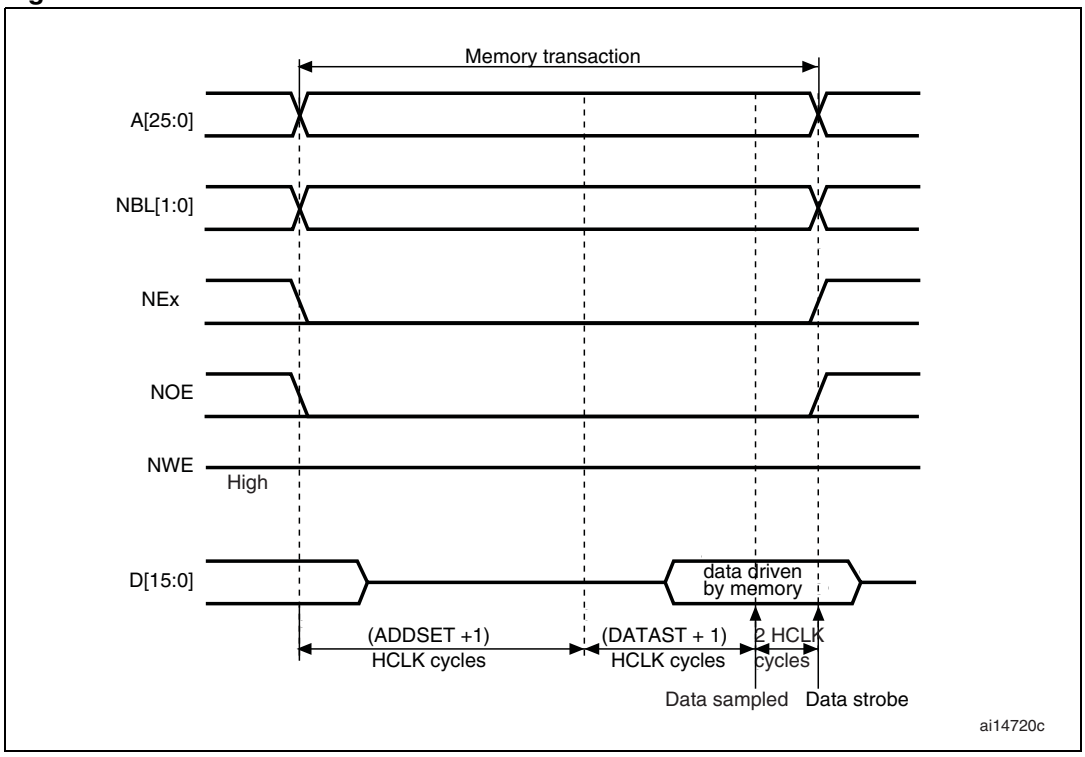
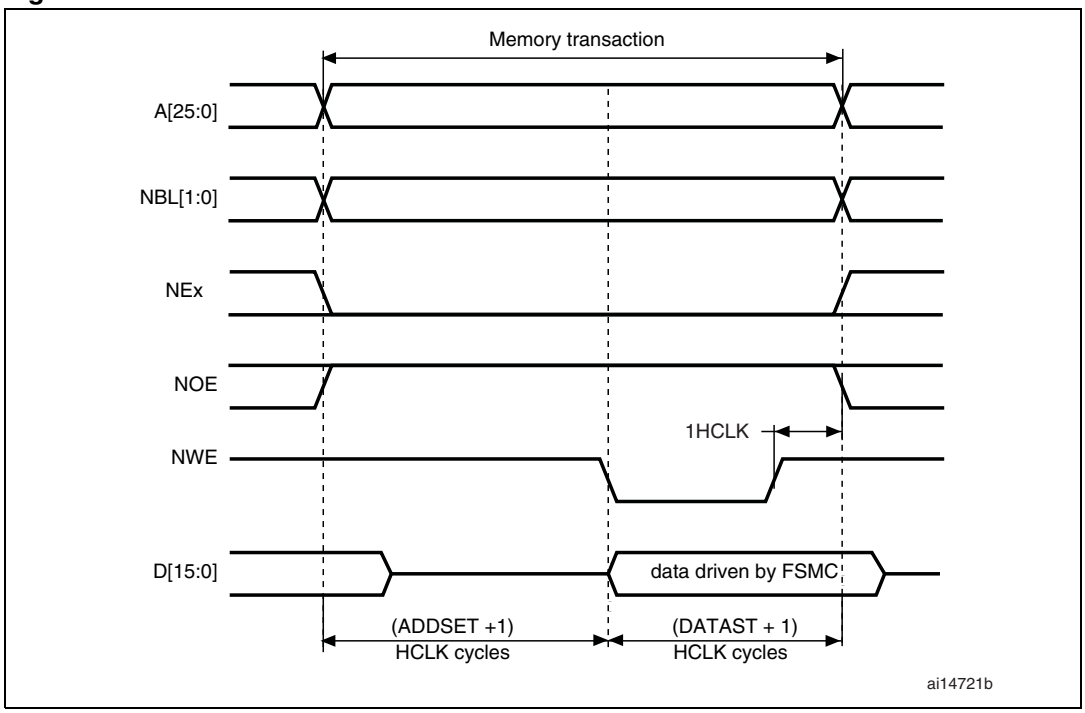


Figure 159. Mode1 write accesses



The one HCLK cycle at the end of the write transaction helps guarantee the address and data hold time after the NWE rising edge. Due to the presence of this one HCLK cycle, the DATAST value must be greater than zero (DATAST > 0).

Table 76. FSMC_BCRx bit fields

Bit number	Bit name	Value to set
31-15		0x0000
14-10		0x0
9	WAITPOL	Meaningful only if bit 15 is 1.
8	BURSTEN	0x0
7		-
6	FACCEN	-
5-4	MWID	As needed
3-2	MTYP	As needed, exclude 10 (NOR Flash).
1	MUXEN	0x0
0	MBKEN	0x1

Table 77. FSMC_TCRx bit fields

Bit number	Bit name	Value to set
31-16		0x0000
15-8	DATAST	Duration of the second access phase (DATAST+1 HCLK cycles) for write accesses, DATAST+3 HCLK cycles for read accesses). This value cannot be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles).

Mode A - SRAM/PSRAM (CRAM) OE toggling

Figure 160. ModeA read accesses

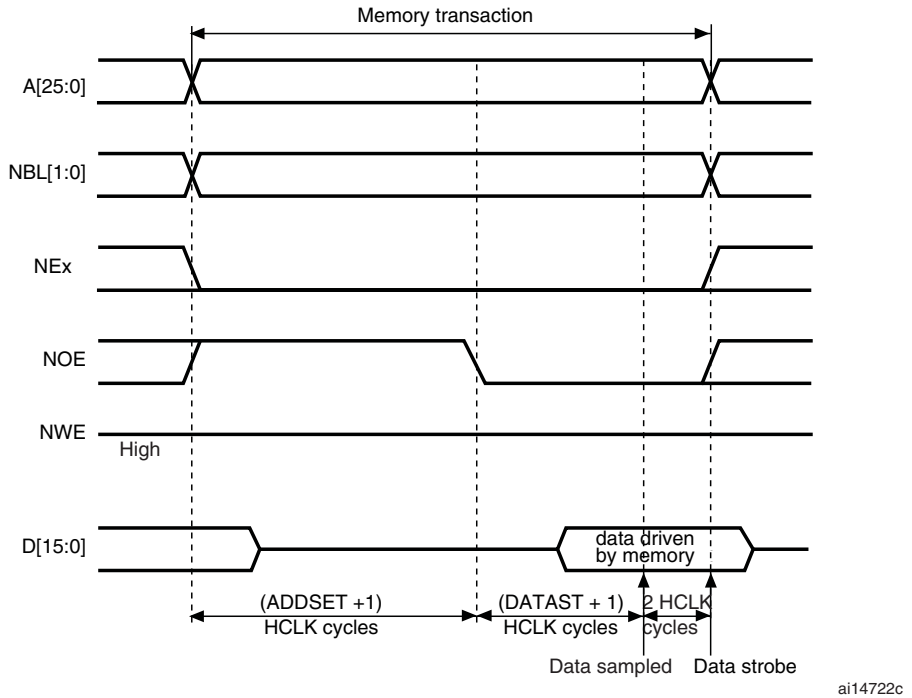
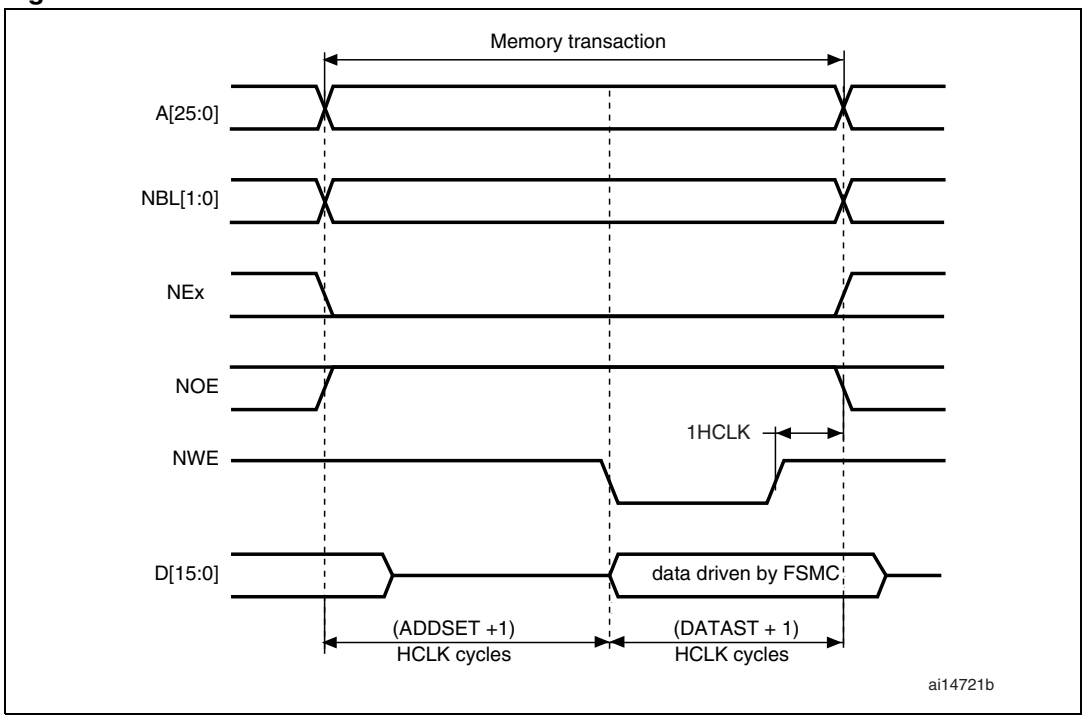


Figure 161. ModeA write accesses



The differences compared with mode1 are the toggling of NOE and the independent read and write timings.

Table 78. FSMC_BCRx bit fields

Bit number	Bit name	Value to set
31-16		0x0000
15		0x0
14	EXTMOD	0x1
13-10		0x0
9	WAITPOL	Meaningful only if bit 15 is 1.
8	BURSTEN	0x0
7		-
6	FACCEN	-
5-4	MWID	As needed
3-2	MTYP	As needed, exclude 10 (NOR Flash).
1	MUXEN	0x0
0	MBKEN	0x1

Table 79. FSMC_TCRx bit fields

Bit number	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x0
27-16		0x000
15-8	DATAST	Duration of the second access phase (DATAST+3 HCLK cycles) in read. This value cannot be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles) in read.

Table 80. FSMC_BWTRx bit fields

Bit number	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x0
27-16		0x000
15-8	DATAST	Duration of the second access phase (DATAST+1 HCLK cycles) in write. This value cannot be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles) in write.

Mode 2/B - NOR Flash

Figure 162. Mode2/B read accesses

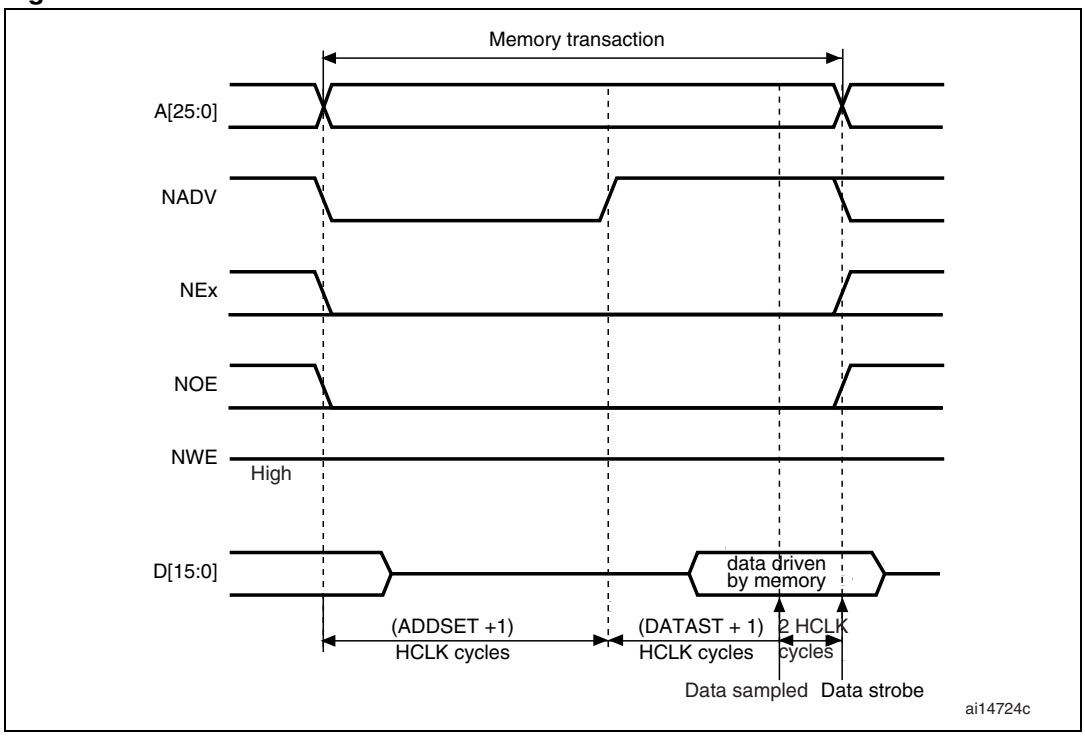


Figure 163. Mode2 write accesses

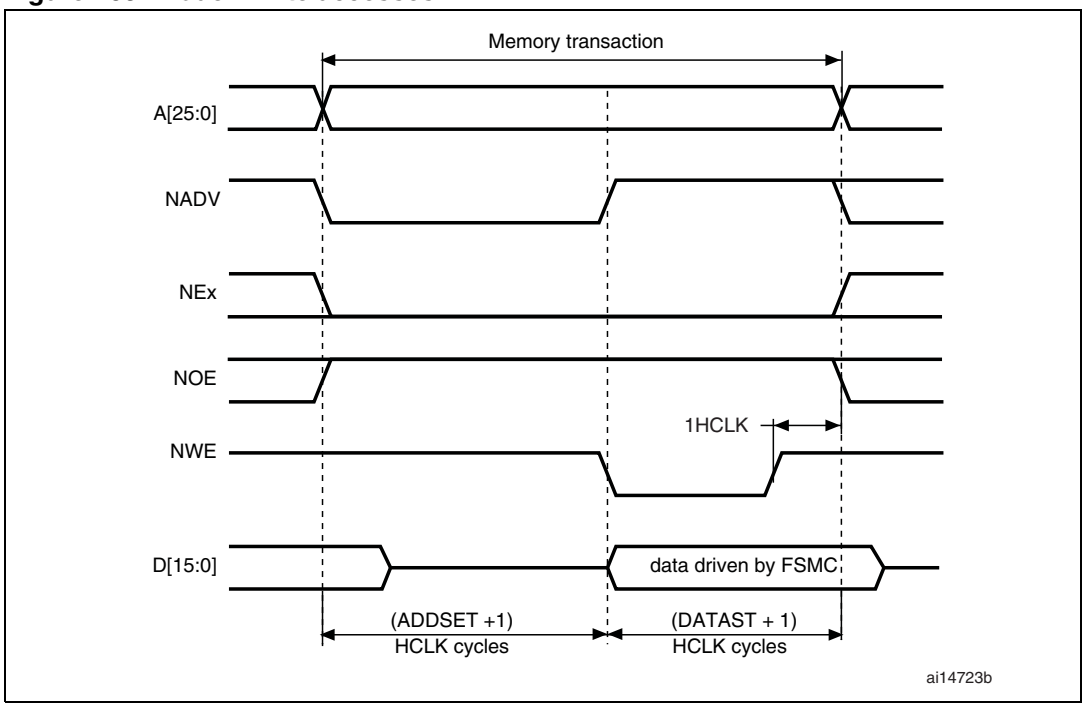
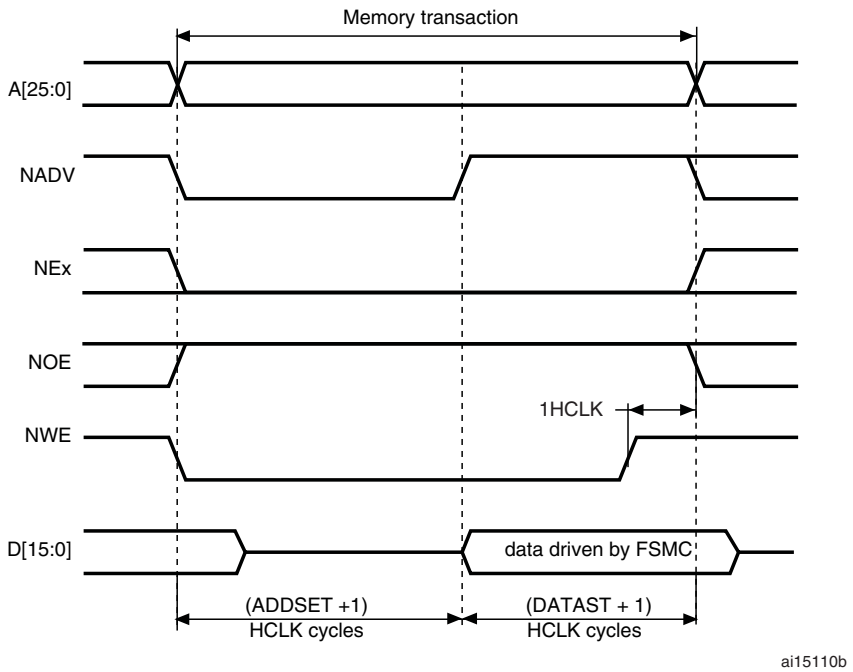


Figure 164. ModeB write accesses



The differences with mode1 are the toggling of NADV and the independent read and write timings when extended mode is set (Mode B).

Table 81. FSMC_BCRx bit fields

Bit number	Bit name	Value to set
31-15		0x0000
14	EXTMOD	0x1 for mode B, 0x0 for mode 2
13-10		0x0
9	WAITPOL	Meaningful only if bit 15 is 1.
8	BURSTEN	0x0
7		-
6	FACCEN	0x1
5-4	MWID	As needed
3-2	MTYP	10 (NOR Flash).
1	MUXEN	0x0
0	MBKEN	0x1

Table 82. FSMC_TCRx bit fields

Bit number	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x1 if extended mode is set
27-16		0x000
15-8	DATAST	Duration of the access second phase (DATAST+3 HCLK cycles) in read. This value can not be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the access first phase (ADDSET+1 HCLK cycles) in read.

Table 83. FSMC_BWTRx bit fields

Bit number	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x1 if extended mode is set
27-16		0x000
15-8	DATAST	Duration of the access second phase (DATAST+1 HCLK cycles) in write. This value can not be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the access first phase (ADDSET+1 HCLK cycles) in write.

Note: The FSMC_BWTRx register is valid only if extended mode is set (mode B), otherwise all its content is don't care.

Mode C - NOR Flash - OE toggling

Figure 165. ModeC read accesses

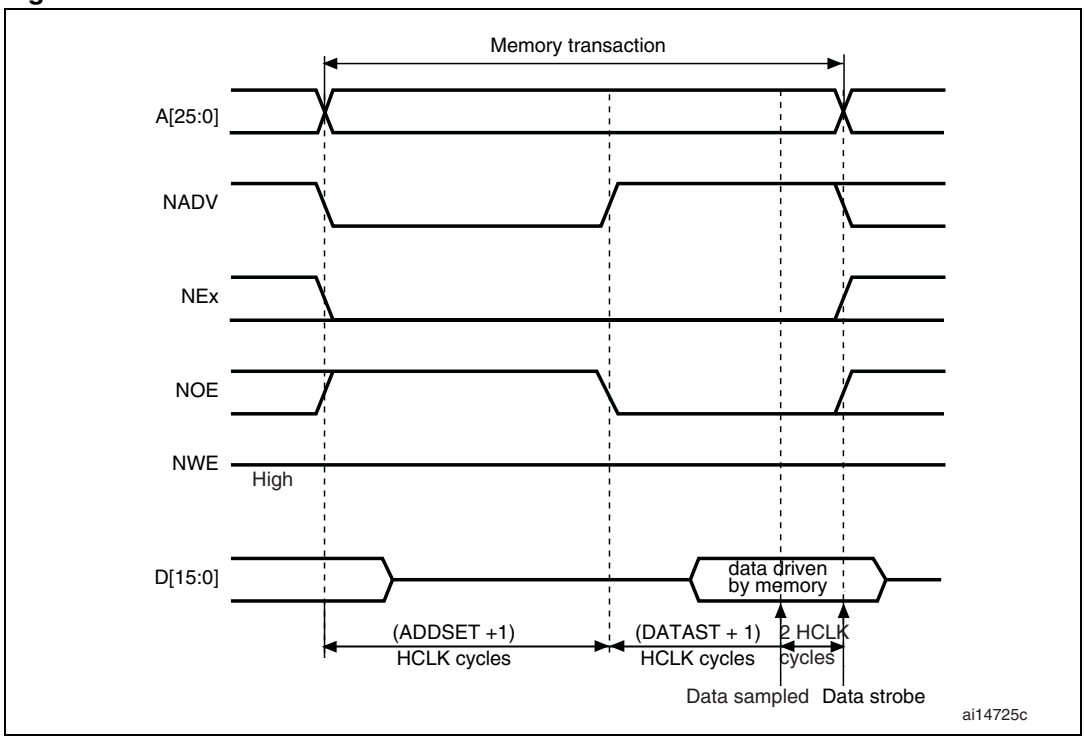
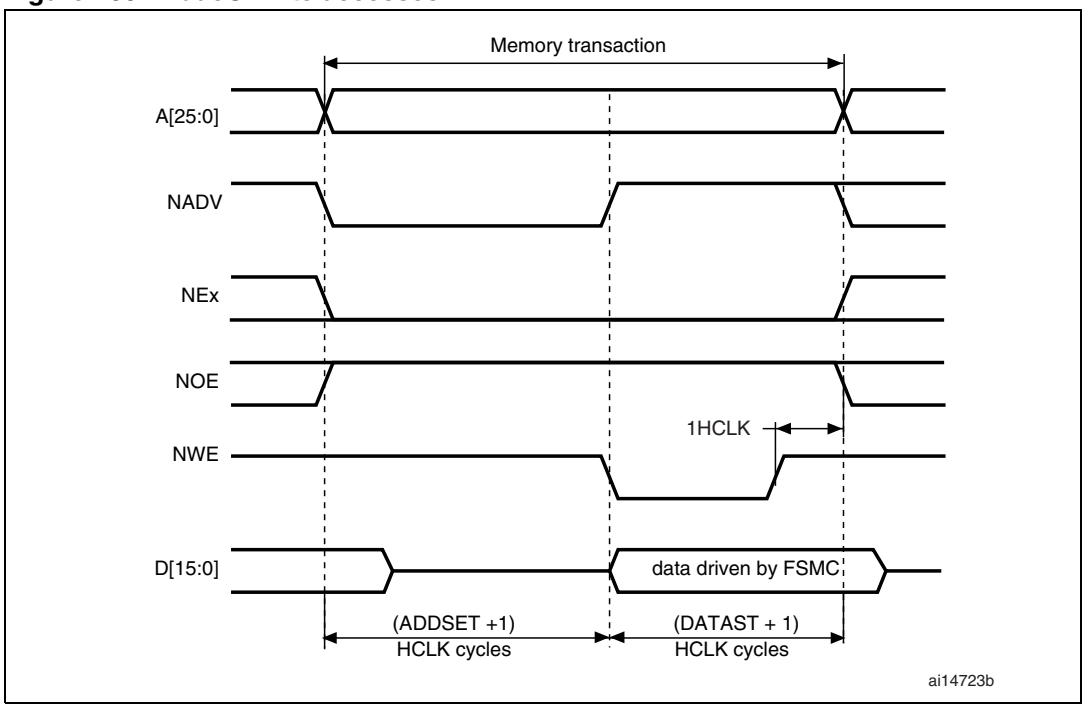


Figure 166. ModeC write accesses



The differences compared with mode1 are the toggling of NOE and NADV and the independent read and write timings.

Table 84. FSMC_BCRx bit fields

Bit No.	Bit name	Value to set
31-15		0x0000
14	EXTMOD	0x1
13-10		0x0
9	WAITPOL	Meaningful only if bit 15 is 1.
8	BURSTEN	0x0
7		-
6	FACCEN	1
5-4	MWID	As needed
3-2	MTYP	0x02 (NOR Flash).
1	MUXEN	0x0
0	MBKEN	0x1

Table 85. FSMC_TCRx bit fields

Bit No.	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x2
27-16		0x000
15-8	DATAST	Duration of the second access phase (DATAST+3 HCLK cycles) in read. This value cannot be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles) in read.

Table 86. FSMC_BWTRx bit fields

Bit No.	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x2
27-16		0x000
15-8	DATAST	Duration of the second access phase (DATAST+1 HCLK cycles) in write. This value cannot be 0 (minimum is 1)
7-4		0x0
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles) in write.

Mode D - asynchronous access with extended address

Figure 167. ModeD read accesses

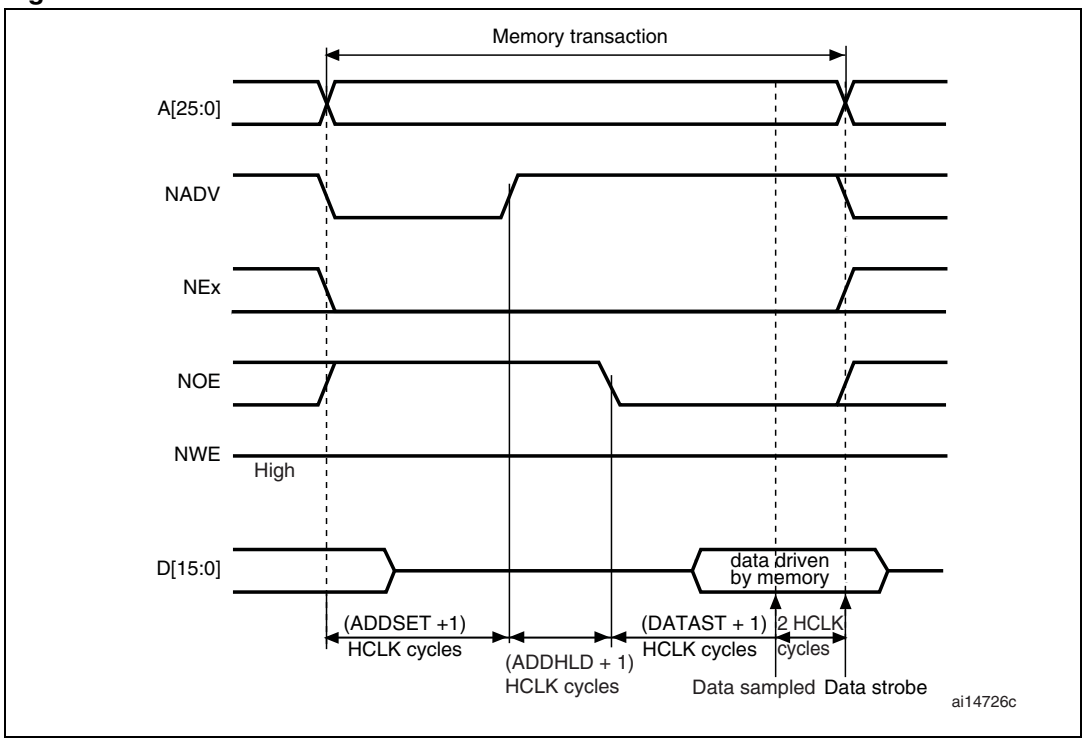
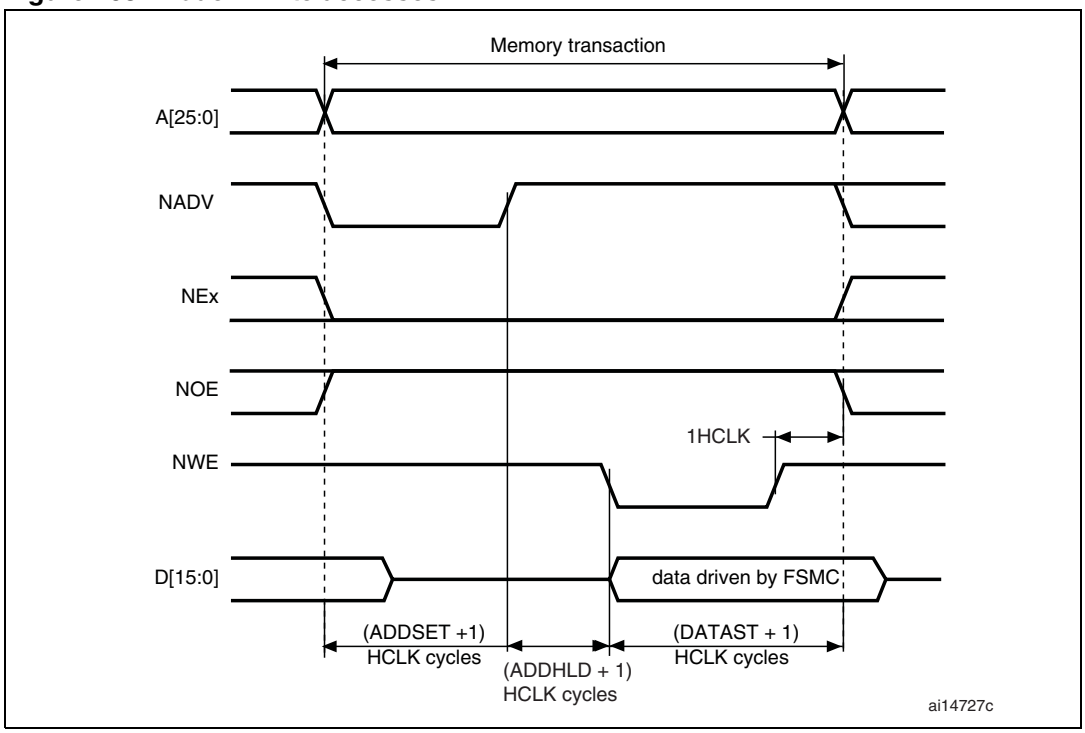


Figure 168. ModeD write accesses



The differences with mode1 are the toggling of NADV, NOE that goes on toggling after NADV changes and the independent read and write timings.

Table 87. FSMC_BCRx bit fields

Bit No.	Bit name	Value to set
31-15		0x0000
14	EXTMOD	0x1
13-10		0x0
9	WAITPOL	Meaningful only if bit 15 is 1.
8	BURSTEN	0x0
7		-
6	FACCEN	Set according to memory support
5-4	MWID	As needed
3-2	MTYP	As needed
1	MUXEN	0x0
0	MBKEN	0x1

Table 88. FSMC_TCRx bit fields

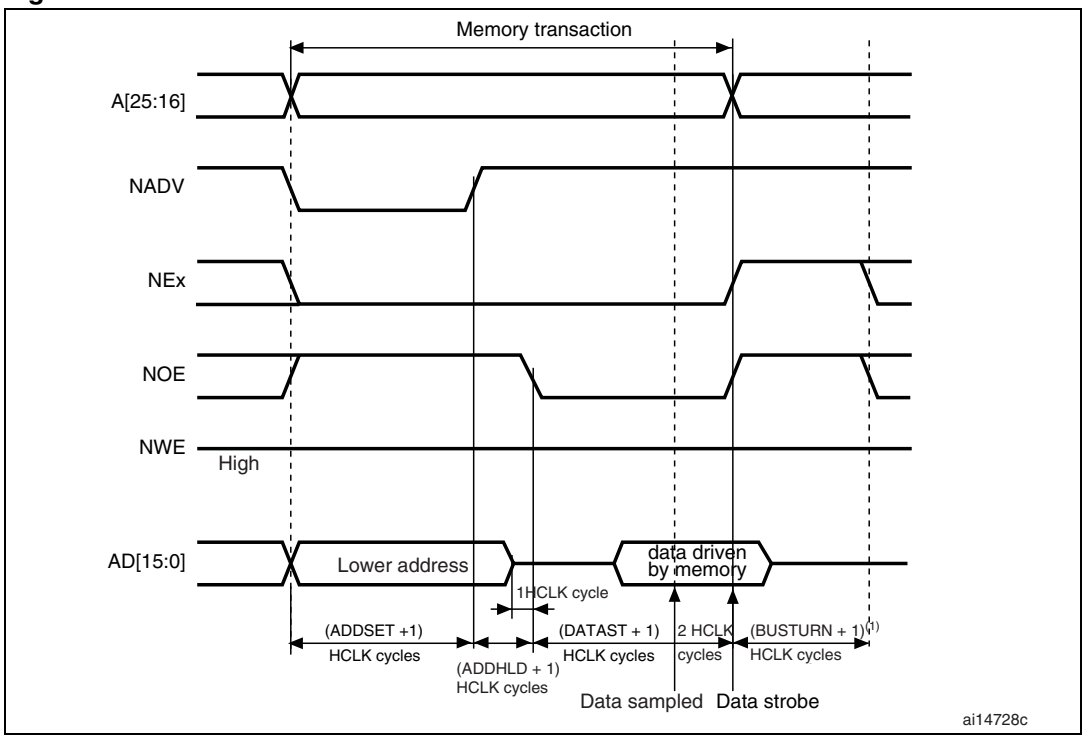
Bit No.	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x2
27-16		0x000
15-8	DATAST	Duration of the second access phase (DATAST+3 HCLK cycles) in read. This value cannot be 0 (minimum is 1)
7-4	ADDHLD	Duration of the middle phase of the read access (ADDHLD+1 HCLK cycles)
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles) in read.

Table 89. FSMC_BWTRx bit fields

Bit No.	Bit name	Value to set
31-30		0x0
29-28	ACCMOD	0x2
27-16		0x000
15-8	DATAST	Duration of the second access phase (DATAST+1 HCLK cycles) in write. This value cannot be 0 (minimum is 1)
7-4	ADDHLD	Duration of the middle phase of the write access (ADDHLD+1 HCLK cycles)
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles) in write.

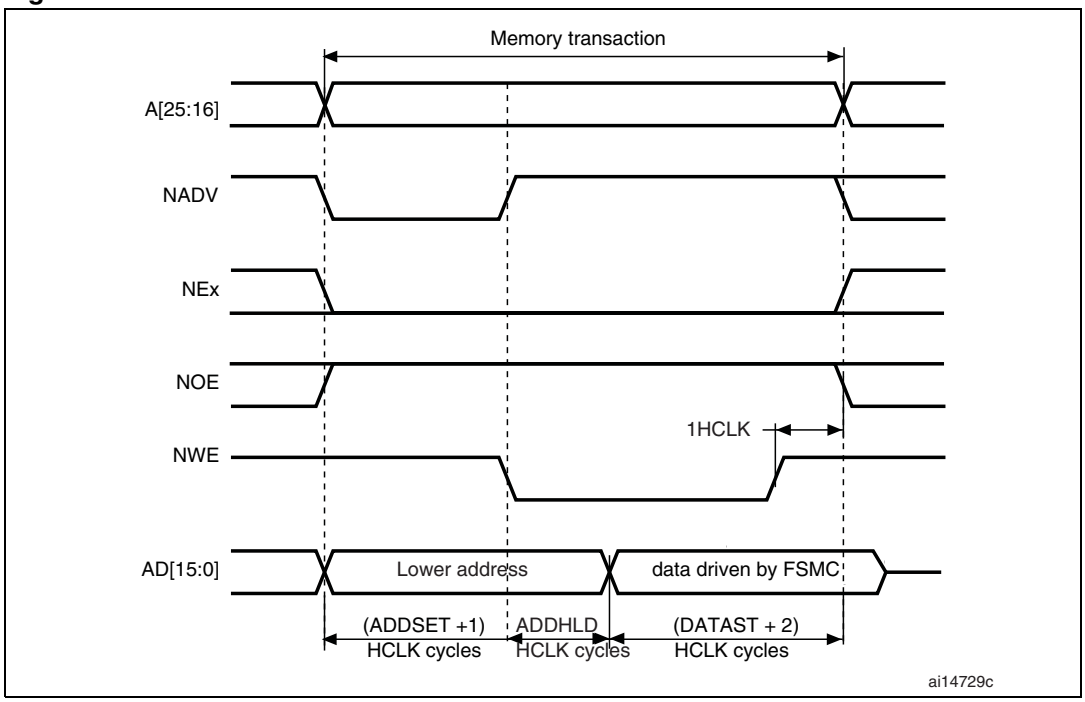
Mode muxed - asynchronous access muxed NOR Flash

Figure 169. Muxed read accesses



1. The bus turnaround delay (BUSTURN + 1) and the delay between side-by-side transactions overlap, so BUSTURN ≤5 has not impact.

Figure 170. Muxed write accesses



The difference with mode D is the drive of the lower address byte(s) on the databus.

Table 90. FSMC_BCRx bit fields

Bit No.	Bit name	Value to set
31-15		0x0000
14	EXTMOD	0x0
13-10		0x0
9	WAITPOL	Meaningful only if bit 15 is 1.
8	BURSTEN	0x0
7		-
6	FACCEN	0x1
5-4	MWID	As needed
3-2	MTYP	0x2 (NOR)
1	MUXEN	0x1
0	MBKEN	0x1

Table 91. FSMC_TCRx bit fields

Bit No.	Bit name	Value to set
31-30		0x0
29-20		
19-16	BUSTURN	Duration of the last phase of the access (BUSTURN+1 HCLK)
15-8	DATAST	Duration of the second access phase (DATAST+3 HCLK cycles for read accesses and DATAST+1 HCLK cycles for write accesses). This value cannot be 0 (minimum is 1)
7-4	ADDHLD	Duration of the middle phase of the access (ADDHLD+1 HCLK cycles). This value cannot be 0 (minimum is 1).
3-0	ADDSET	Duration of the first access phase (ADDSET+1 HCLK cycles).

18.5.5 Synchronous burst read

The memory clock, CLK, is a submultiple of HCLK according to the value of parameter CLKDIV.

NOR Flash memories specify a minimum time from NADV assertion to CLK high. To meet this constraint, the FSMC does not issue the clock to the memory during the first internal clock cycle of the synchronous access (before NADV assertion). This guarantees that the rising edge of the memory clock occurs *in the middle* of the NADV low pulse.

Data latency versus NOR Flash latency

The data latency is the number of cycles to wait before sampling the data. The DATLAT value must be consistent with the latency value specified in the NOR Flash configuration register. The FSMC does not include the clock cycle when NADV is low in the data latency count.

Caution: Some NOR Flash memories include the NADV Low cycle in the data latency count, so the exact relation between the NOR Flash latency and the FMSC DATLAT parameter can be either of:

- NOR Flash latency = DATLAT + 2
- NOR Flash latency = DATLAT + 3

Some recent memories assert NWAIT during the latency phase. In such cases DATLAT can be set to its minimum value. As a result, the FSMC samples the data and waits long enough to evaluate if the data are valid. Thus the FSMC detects when the memory exits latency and real data are taken.

Other memories do not assert NWAIT during latency. In this case the latency must be set correctly for both the FSMC and the memory, otherwise invalid data are mistaken for good data, or valid data are lost in the initial phase of the memory access.

Single-burst transfer

When the selected bank is configured in synchronous burst mode, if an AHB single-burst transaction is requested, the FSMC performs a burst read of length 1 (if the AHB transfer is 16-bit), or length 2 (if the AHB transfer is 32-bit, thus split into two 16-bit accesses) and de-assert the chip select signal when the last data is strobed.

Clearly, such a transfer is not the most efficient in terms of cycles (compared to an asynchronous read). Nevertheless, a random asynchronous access would first require to re-program the memory access mode, which would altogether last longer.

Wait management

For synchronous burst NOR Flash, NWAIT is evaluated after the programmed latency period, (DATALAT+1) CLK clock cycles.

If NWAIT is sensed active (low level when WAITPOL = 0, high level when WAITPOL = 1), wait states are inserted until NWAIT is sensed inactive (high level when WAITPOL = 0, low level when WAITPOL = 1).

When NWAIT is inactive, the data is considered valid either immediately (bit WAITCFG = 1) or on the next clock edge (bit WAITCFG = 0).

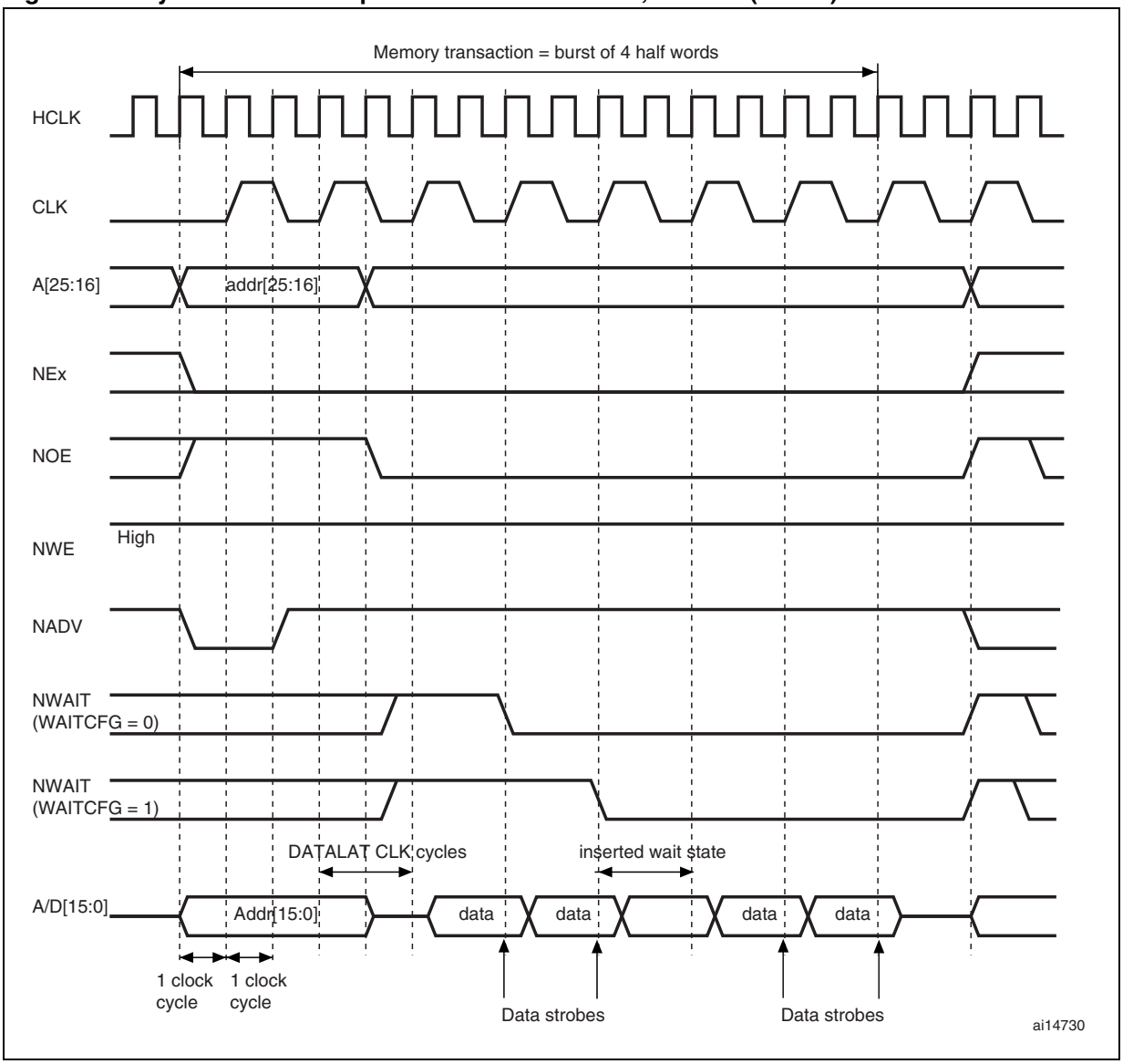
During wait-state insertion via the NWAIT signal, the controller continues to send clock pulses to the memory, keeping the chip select and output enable signals valid, and does not consider the data valid.

There are two timing configurations for the NOR Flash NWAIT signal in burst mode:

- Flash memory asserts the NWAIT signal one data cycle before the wait state (default after reset)
- Flash memory asserts the NWAIT signal during the wait state

These two NOR Flash wait state configurations are supported by the FSMC, individually for each chip select, thanks to the WAITCFG bit in the FSMC_BCRx registers (x = 0..3).

Figure 171. Synchronous multiplexed read mode - NOR, PSRAM (CRAM)



1. Byte lane outputs BL are not shown; for NOR access, they are held high, and, for PSRAM (CRAM) access, they are held low.

Table 92. FSMC_BCRx bit fields

Bit No.	Bit name	Value to set
31-20		0x0000
19	CBURSTRW	No effect on synchronous read
18-15		0x0
14	EXTMOD	0x0
13	WAITEN	When high, the first data after latency period is taken as always valid, regardless of the wait from memory value
12	WREN	no effect on synchronous read

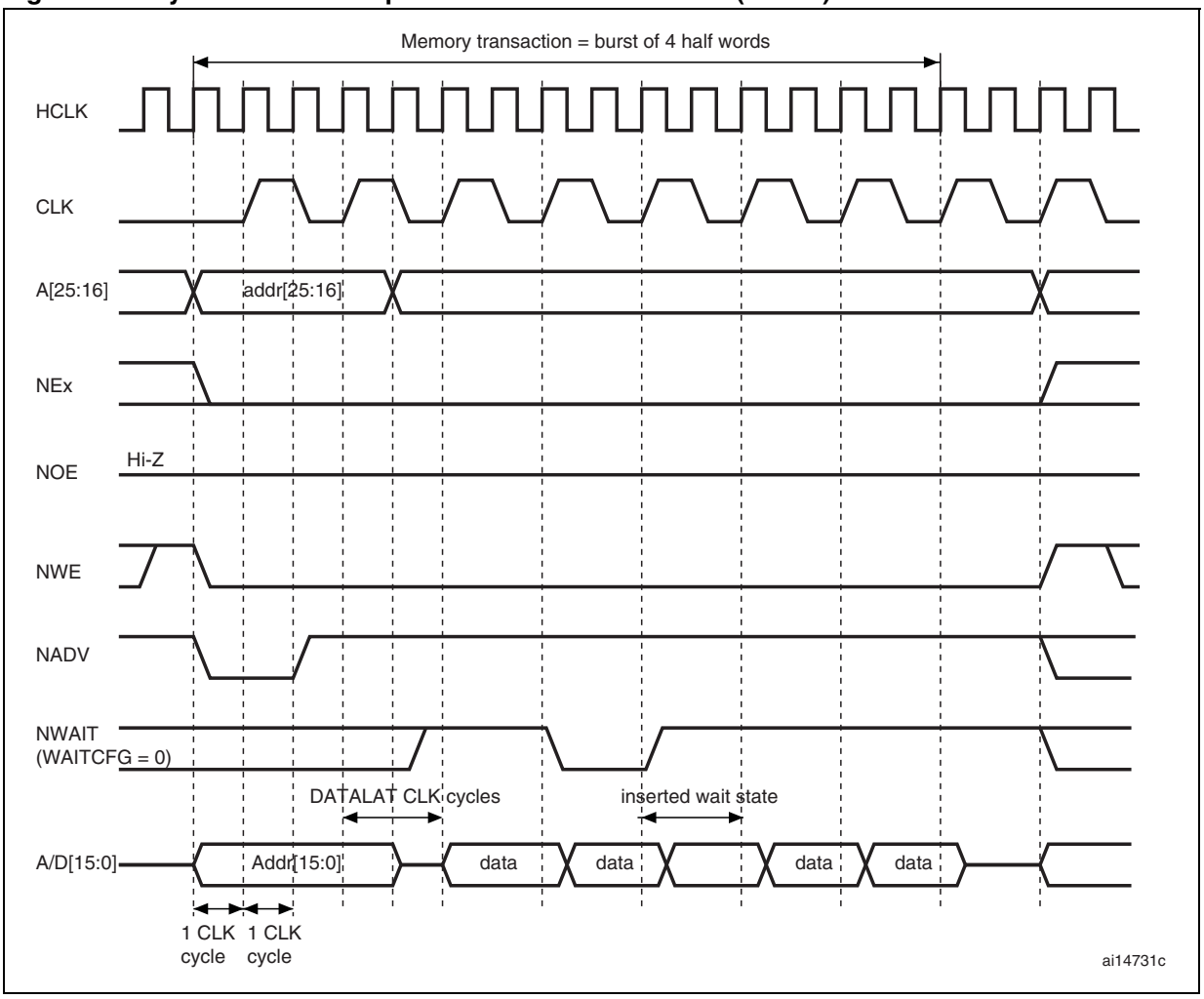
Table 92. FSMC_BCRx bit fields (continued)

Bit No.	Bit name	Value to set
11	WAITCFG	to be set according to memory
10	WRAPMOD	to be set according to memory
9	WAITPOL	to be set according to memory
8	BURSTEN	0x1
7	FWPRLVL	Set to protect memory from accidental write access
6	FACCEN	Set according to memory support
5-4	MWID	As needed
3-2	MTYP	0x1 or 0x2
1	MUXEN	As needed
0	MBKEN	0x1

Table 93. FSMC_TCRx bit fields

Bit No.	Bit name	Value to set
27-24	DATLAT	Data latency
23-20	CLKDIV	0x0 to get CLK = HCLK (not supported) 0x1 to get CLK = 2 × HCLK
19-16	BUSTURN	no effect
15-8	DATAST	no effect
7-4	ADDHLD	no effect
3-0	ADDSET	no effect

Figure 172. Synchronous multiplexed write mode - PSRAM (CRAM)



1. Memory must issue NWAIT signal one cycle in advance, accordingly WAITCFG must be programmed to 0.
2. Byte Lane (NBL) outputs are not shown, they are held low while NEx is active.

Table 94. FSMC_BCRx bit fields

Bit No.	Bit name	Value to set
31-20		0x0000
19	CBURSTRW	0x1
18-15		0x0
14	EXTMOD	0x0
13	WAITEN	When high, the first data after latency period is taken as always valid, regardless of the wait from memory value
12	WREN	no effect on synchronous read
11	WAITCFG	0x0
10	WRAPMOD	to be set according to memory
9	WAITPOL	to be set according to memory
8	BURSTEN	no effect on synchronous write
7	FWPRLVL	Set to protect memory from accidental writes
6	FACCEN	Set according to memory support
5-4	MWID	As needed
3-2	MTYP	0x1
1	MUXEN	As needed
0	MBKEN	0x1

Table 95. FSMC_TCRx bit fields

Bit No.	Bit name	Value to set
31-30	-	0x0
27-24	DATLAT	Data latency
23-20	CLKDIV	0 to get CLK = HCLK (not supported) 1 to get CLK = 2 × HCLK
19-16	BUSTURN	No effect
15-8	DATAST	No effect
7-4	ADDHLD	No effect
3-0	ADDSET	No effect

18.5.6 NOR/PSRAM controller registers

SRAM/NOR-Flash chip-select control registers 1..4 (FSMC_BCR1..4)

Address offset: 0xA000 0000 + 8 * (x – 1), x = 1...4

Reset value: 0x0000 30DX

This register contains the control information of each memory bank, used for SRAMs, ROMs and asynchronous or burst NOR Flash memories.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												CBURSTRW	Reserved				EXTMOD	WAITEN	WREN	WAITCFG	WRAPMOD	WAITPOL	BURSTEN	Reserved	FACCEN	MWID		MTYP		MUXEN	MBKEN
												rw					rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bit 19 **CBURSTRW:** Write burst enable.

For Cellular RAM, the bit enables synchronous burst protocol during write operations. For Flash memory access in burst mode, this bit enables/disables the wait state insertion via the NWAIT signal. The enable bit for the synchronous burst protocol during read access is the BURSTEN bit in the FSMC_BCRx register.

0: Write operations are always performed in asynchronous mode

1: Write operations are performed in synchronous mode.

Bit 15 Reserved.

Bit 14 **EXTMOD:** Extended mode enable.

This bit enables the FSMC to program inside the FSMC_BWTR register, so it allows different timings for read and write.

0: values inside FSMC_BWTR register are not taken into account (default after reset)

1: values inside FSMC_BWTR register are taken into account

Bit 13 **WAITEN:** Wait enable bit.

For Flash memory access in burst mode, this bit enables/disables wait-state insertion via the NWAIT signal:

0: NWAIT signal is disabled (its level not taken into account, no wait state inserted after the programmed Flash latency period)

1: NWAIT signal is enabled (its level is taken into account after the programmed Flash latency period to insert wait states if asserted) (default after reset)

Bit 12 **WREN:** Write enable bit.

This bit indicates whether write operations are enabled/disabled in the bank by the FSMC:

0: Write operations are disabled in the bank by the FSMC, an AHB error is reported,

1: Write operations are enabled for the bank by the FSMC (default after reset).

Bit 11 **WAITCFG:** Wait timing configuration.

For memory access in burst mode, the NWAIT signal indicates whether the data from the memory are valid or if a wait state must be inserted. This configuration bit determines if NWAIT is asserted by the memory one clock cycle before the wait state or during the wait state:

0: NWAIT signal is active one data cycle before wait state (default after reset),

1: NWAIT signal is active during wait state (not for Cellular RAM).

Bit 10 **WRAPMOD:** Wrapped burst mode support.

Defines whether the controller will or not split an AHB burst wrap access into two linear accesses. Valid only when accessing memories in burst mode

0: Direct wrapped burst is not enabled (default after reset),

1: Direct wrapped burst is enabled.

Bit 9 **WAITPOL:** Wait signal polarity bit.

Defines the polarity of the wait signal from memory. Valid only when accessing the memory in burst mode:

0: NWAIT active low (default after reset),

1: NWAIT active high.

Bit 8 **BURSTEN:** Burst enable bit.

Enables the burst access mode for the memory. Valid only with synchronous burst memories:

0: Burst access mode disabled (default after reset)

1: Burst access mode enable

Bit 7 Reserved.

Bit 6 **FACCEN:** Flash access enable

Enables NOR Flash memory access operations.

0: Corresponding NOR Flash memory access is disabled

1: Corresponding NOR Flash memory access is enabled (default after reset)

Bits 5:4 **MWID:** Memory databus width.

Defines the external memory device width, valid for all type of memories.

00: 8 bits,

01: 16 bits (default after reset),

10: reserved, do not use,

11: reserved, do not use.

Bits 3:2 **MTYP:** Memory type.

Defines the type of external memory attached to the corresponding memory bank:

00: SRAM, ROM (default after reset for Bank 2...4)

01: PSRAM (Cellular RAM: CRAM)

10: NOR Flash (default after reset for Bank 1)

11: reserved

Bit 1 **MUXEN:** Address/data multiplexing enable bit.

When this bit is set, the address and data values are multiplexed on the databus, valid only with NOR and PSRAM memories:

0: Address/Data nonmultiplexed

1: Address/Data multiplexed on databus (default after reset)

Bit 0 **MBKEN:** Memory bank enable bit.

Enables the memory bank. After reset Bank1 is enabled, all others are disabled. Accessing a disabled bank causes an ERROR on AHB bus.

0: Corresponding memory bank is disabled

1: Corresponding memory bank is enabled

SRAM/NOR-Flash chip-select timing registers 1..4 (FSMC_BTR1..4)

Address offset: 0xA000 0000 + 0x04 + 8 * (x – 1), x = 1..4

Reset value: 0x0FFF FFFF

This register contains the control information of each memory bank, used for SRAMs, ROMs and NOR Flash memories. If the EXTMOD bit is set in the FSMC_BCRx register, then this register is partitioned for write and read access, that is, 2 registers are available: one to configure read accesses (this register) and one to configure write accesses (FSMC_BWTRx registers).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		ACCMOD		DATLAT				CLKDIV				BUSTURN				DATAST								ADDHLD				ADDSET			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 29:28 **ACCMOD:** Access mode.

Specifies the asynchronous access modes as shown in the timing diagrams. These bits are taken into account only when the EXTMOD bit in the FSMC_BCRx register is 1.

00: access mode A

01: access mode B

10: access mode C

11: access mode D

Bits 27:24 **DATLAT** (see note below bit descriptions)**:** Data latency (for synchronous burst NOR Flash).

For NOR Flash with synchronous burst mode enabled, defines the number of memory clock cycles (+2) to issue to the memory before getting the first data:

This timing parameter is not expressed in HCLK periods, but in Flash clock (CLK) periods. In asynchronous NOR Flash, SRAM or ROM accesses, this value is don't care. In case of CRAM, this field must be set to 0

0000: Data latency of 2 CLK clock cycles for first burst access

1111: Data latency of 17 CLK clock cycles for first burst access (default value after reset)

Bits 23:20 **CLKDIV:** Clock divide ratio (for CLK signal).

Defines the period of CLK clock output signal, expressed in number of HCLK cycles:

0000: Reserved

0001: CLK period = 2 × HCLK periods

0010: CLK period = 3 × HCLK periods

1111: CLK period = 16 × HCLK periods (default value after reset)

In asynchronous NOR Flash, SRAM or ROM accesses, this value is don't care.

Bits 19:16 **BUSTURN:** Bus turnaround phase duration.

These bits are written by software to introduce the bus turnaround delay after a read access (only from multiplexed NOR Flash memory) to avoid bus contention if the controller needs to drive addresses on the databus for the next side-by-side transaction. BUSTURN can be set to the minimum if the memory system does not include multiplexed memories or if the slowest memory does not take more than 6 HCLK clock cycles to put the databus in Hi-Z state:

0000: bus turnaround duration = 1 × HCLK clock cycle

...

1111: bus turnaround duration = 16 × HCLK clock cycles (default value after reset)

Bits 15:8 **DATAST:** Data-phase duration.

These bits are written by software to define the duration of the data phase (refer to *Figure 156* to *Figure 168*), used in SRAMs, ROMs and asynchronous multiplexed NOR Flash accesses:

0000 0000: DATAST phase duration = 1 × HCLK clock cycle

...

0000_1111: DATAST phase duration = 16 × HCLK clock cycles (default value after reset)

Bits 7:4 **ADDHLD:** Address-hold phase duration.

These bits are written by software to define the duration of the *address hold* phase (refer to *Figure 165* to *Figure 168*), used in SRAMs, ROMs and asynchronous multiplexed NOR Flash accesses:

0000: ADDHLD phase duration = 1 × HCLK clock cycle

...

1111: ADDHLD phase duration = 16 × HCLK clock cycles (default value after reset)

Note: In synchronous accesses, this value is not used, the address hold phase is always 1 memory clock period duration.

Bits 3:0 **ADDSET:** Address setup phase duration.

These bits are written by software to define the duration of the *address setup* phase in HCLK cycles (refer to *Figure 165* to *Figure 168*), used in SRAMs, ROMs and asynchronous multiplexed NOR Flash:

0000: ADDSET phase duration = 1 × HCLK clock cycle

...

1111: ADDSET phase duration = 16 × HCLK clock cycles (default value after reset)

Note: In synchronous accesses, this value is not used, the address hold phase is always 1 memory clock period duration.

Note: PSRAMs (CRAMs) have a variable latency due to internal refresh. Therefore these memories issue the NWAIT signal during the whole latency phase to prolong the latency as needed.
With PSRAMs (CRAMs) the filed DATLAT must be set to 0, so that the FSMC exits its latency phase soon and starts sampling NWAIT from memory, then starts to read or write when the memory is ready.
This method can be used also with the latest generation of synchronous Flash memories that issue the NWAIT signal, unlike older Flash memories (check the datasheet of the specific Flash memory being used).

SRAM/NOR-Flash write timing registers 1..4 (FSMC_BWTR1..4)

Address offset: 0xA000 0000 + 0x104 + 8 * (x – 1), x = 1...4

Reset value: 0x0FFF FFFF

This register contains the control information of each memory bank, used for SRAMs, ROMs and NOR Flash memories. When the EXTMOD bit is set in the FSMC_BCRx register, then this register is active for write access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.		ACCMOD		DATLAT				CLKDIV				Reserved				DATAST								ADDHLD				ADDSET			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 29:28 **ACCMOD:** Access mode.

Specifies the asynchronous access modes as shown in the next timing diagrams.These bits are taken into account only when the EXTMOD bit in the FSMC_BCRx register is 1.

00: access mode A

01: access mode B

10: access mode C

11: access mode D

Bits 27:24 **DATLAT:** Data latency (for synchronous burst NOR Flash).

For NOR Flash with Synchronous burst mode enabled, defines the number of memory clock cycles (+2) to issue to the memory before getting the first data:

0000: (0x0) Data latency of 2 CLK clock cycles for first burst access

...

1111: (0xF) Data latency of 17 CLK clock cycles for first burst access (default value after reset)

*Note: This timing parameter is not expressed in HCLK periods, but in Flash clock (**CLK**) periods*

Note: In asynchronous NOR Flash, SRAM or ROM accesses, this value is don't care.

Note: In case of CRAM, this field must be set to 0

Bits 23:20 **CLKDIV:** Clock divide ratio (for CLK signal).

Defines the period of CLK clock output signal, expressed in number of HCLK cycles:

0000: Reserved

0001 CLK period = 2 × HCLK periods

0010 CLK period = 3 × HCLK periods

1111: CLK period = 16 × HCLK periods (default value after reset)

In asynchronous NOR Flash, SRAM or ROM accesses, this value is don't care.

Bits 19:16 Reserved

Bits 15:8 **DATAST:** Data-phase duration.

These bits are written by software to define the duration of the data phase (refer to *Figure 156* to *Figure 168*), used in SRAMs, ROMs and asynchronous multiplexed NOR Flash accesses:

0000 0000: DATAST phase duration = 1 × HCLK clock cycle

...

0000_1111: DATAST phase duration = 16 × HCLK clock cycles (default value after reset)

Bits 7:4 **ADDHLD:** Address-hold phase duration.

These bits are written by software to define the duration of the *address hold* phase (refer to *Figure 165* to *Figure 168*), used in SRAMs, ROMs and asynchronous multiplexed NOR Flash accesses:

0000: ADDHLD phase duration = 1 × HCLK clock cycle -->

...

1111: ADDHLD phase duration = 16 × HCLK clock cycles (default value after reset)

Note: In synchronous NOR Flash accesses, this value is not used, the address hold phase is always 1 Flash clock period duration.

Bits 3:0 **ADDSET:** Address setup phase duration.

These bits are written by software to define the duration of the *address setup* phase in HCLK cycles (refer to *Figure 165* to *Figure 168*), used in SRAMs, ROMs and asynchronous multiplexed NOR Flash:

0000: ADDSET phase duration = 1 × HCLK clock cycle

...

1111: ADDSET phase duration = 16 × HCLK clock cycles (default value after reset)

Note: In synchronous NOR Flash accesses, this value is not used, the address hold phase is always 1 Flash clock period duration.

18.6 NAND Flash/PC Card controller

The FSMC generates the appropriate signal timings to drive the following types of device:

- NAND Flash
  - 8-bit
  - 16-bit
- 16-bit PC Card compatible devices

The NAND/PC Card controller can control three external banks. Bank 2 and bank 3 support NAND Flash devices. Bank 4 supports PC Card devices.

Each bank is configured by means of dedicated registers (*Section 18.6.7*). The programmable memory parameters include access timings (shown in *Table 96*) and ECC configuration.

Table 96. Programmable NAND/PC Card access parameters

Parameter	Function	Access mode	Unit	Min.	Max.
Memory setup time	Number of clock cycles (HCLK) to set up the address before the command assertion	Read/Write	AHB clock cycle (HCLK)	1	256
Memory wait	Minimum duration (HCLK clock cycles) of the command assertion	Read/Write	AHB clock cycle (HCLK)	1	256
Memory hold	Number of clock cycles (HCLK) to hold the address (and the data in case of a write access) after the command de-assertion	Read/Write	AHB clock cycle (HCLK)	1	255
Memory databus high-Z	Number of clock cycles (HCLK) during which the databus is kept in high-Z state after the start of a write access	Write	AHB clock cycle (HCLK)	0	255

18.6.1 External memory interface signals

The following tables list the signals that are typically used to interface NAND Flash and PC Card.

Caution: When using a PC Card or a CompactFlash in I/O mode, the NIOS16 input pin must remain at ground level during the whole operation, otherwise the FSMC may not operate properly. This means that the NIOS16 input pin must *not* be connected to the card, but directly to ground (only 16-bit accesses are allowed).

Note: Prefix "N". specifies the associated signal as active low.

8-bit NAND Flash

Table 97. 8-bit NAND Flash

FSMC signal name	I/O	Function
A[17]	O	NAND Flash address latch enable (ALE) signal
A[16]	O	NAND Flash command latch enable (CLE) signal
D[7:0]	I/O	8-bit multiplexed, bidirectional address/data bus
NCE[x]	O	Chip select, x = 2, 3
NOE(= NRE)	O	Output enable (memory signal name: read enable, NRE)
NWE	O	Write enable
NWAIT/INT[3:2]	I	NAND Flash ready/busy input signal to the FSMC

There is no theoretical capacity limitation as the FSMC can manage as many address cycles as needed.

16-bit NAND Flash

Table 98. 16-bit NAND Flash

FSMC signal name	I/O	Function
A[17]	O	NAND Flash address latch enable (ALE) signal
A[16]	O	NAND Flash command latch enable (CLE) signal
D[15:0]	I/O	16-bit multiplexed, bidirectional address/data bus
NCE[x]	O	Chip select, x = 2, 3
NOE(= NRE)	O	Output enable (memory signal name: read enable, NRE)
NWE	O	Write enable
NWAIT/INT[3:2]	I	NAND Flash ready/busy input signal to the FSMC

There is no theoretical capacity limitation as the FSMC can manage as many address cycles as needed.

Table 99. 16-bit PC Card

FSMC signal name	I/O	Function
A[10:0]	O	Address bus
NIOS16	I	Data transfer width in I/O space (16-bit transfer only)
NIORD	O	Output enable for I/O space
NIOWR	O	Write enable for I/O space
NREG	O	Register signal indicating if access is in Common or Attribute space
D[15:0]	I/O	Bidirectional databus
NCE4_1	O	Chip select 1
NCE4_2	O	Chip select 2 (indicates if access is 16-bit or 8-bit)
NOE	O	Output enable
NWE	O	Write enable
NWAIT	I	PC Card wait input signal to the FSMC (memory signal name IORDY)
INTR	I	PC Card interrupt to the FSMC (only for PC Cards that can generate an interrupt)
CD	I	PC Card presence detection

18.6.2 NAND Flash / PC Card supported memories and transactions

Table 100 below shows the supported devices, access modes and transactions. Transactions not allowed (or not supported) by the NAND Flash / PC Card controller appear in gray.

Table 100. Supported memories and transactions

Device	Mode	R/W	AHB data size	Memory data size	Allowed/ not allowed	Comments
NAND 8-bit	Asynchronous	R	8	8	Y	
	Asynchronous	W	8	8	Y	
	Asynchronous	R	16	8	Y	Split into 2 FSMC accesses
	Asynchronous	W	16	8	Y	Split into 2 FSMC accesses
	Asynchronous	R	32	8	Y	Split into 4 FSMC accesses
	Asynchronous	W	32	8	Y	Split into 4 FSMC accesses
NAND 16-bit	Asynchronous	R	8	16	Y	
	Asynchronous	W	8	16	N	
	Asynchronous	R	16	16	Y	
	Asynchronous	W	16	16	Y	
	Asynchronous	R	32	16	Y	Split into 2 FSMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FSMC accesses

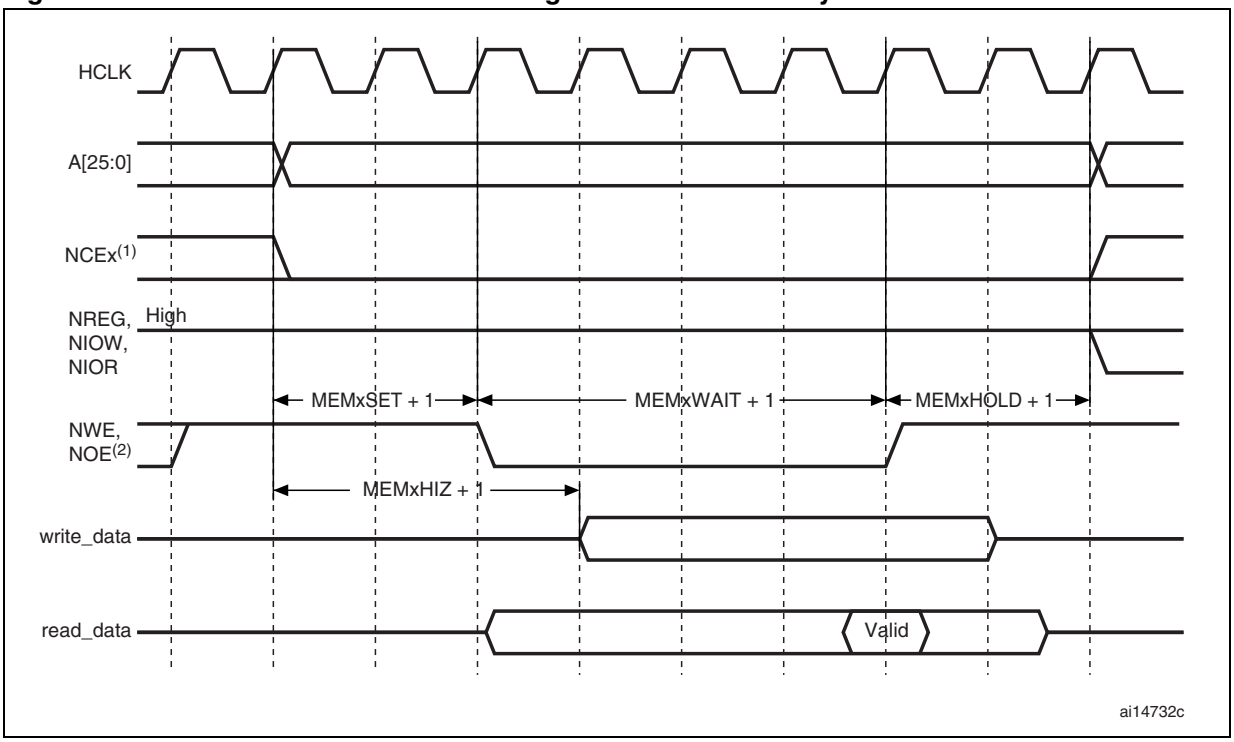
18.6.3 Timing diagrams for NAND, ATA and PC Card

Each PC Card/CompactFlash and NAND Flash memory bank is managed through a set of registers:

- Control register: FSMC_PCRx
- Interrupt status register: FSMC_SRx
- ECC register: FSMC_ECCRx
- Timing register for Common memory space: FSMC_PMEMx
- Timing register for Attribute memory space: FSMC_PATTx
- Timing register for I/O space: FSMC_PIOx

Each timing configuration register contains three parameters used to define number of HCLK cycles for the three phases of any PC Card/CompactFlash or NAND Flash access, plus one parameter that defines the timing for starting driving the databus in the case of a write. *Figure 173* shows the timing parameter definitions for common memory accesses, knowing that Attribute and I/O (only for PC Card) memory space access timings are similar.

Figure 173. NAND/PC Card controller timing for common memory access



1. NCEx goes low as soon as NAND access is requested and remains low until a different memory bank is accessed.
2. NOE remains high (inactive) during write access. NWE remains high (inactive) during read access.

18.6.4 NAND Flash operations

The command latch enable (CLE) and address latch enable (ALE) signals of the NAND Flash device are driven by some address signals of the FSMC controller. This means that to send a command or an address to the NAND Flash memory, the CPU has to perform a write to a certain address in its memory space.

A typical page read operation from the NAND Flash device is as follows:

1. Program and enable the corresponding memory bank by configuring the FSMC_PCRx and FSMC_PMEMx (and for some devices, FSMC_PATTx, see *Section 18.6.5: NAND Flash pre-wait functionality on page 395*) registers according to the characteristics of the NAND Flash (PWID bits for the databus width of the NAND Flash, PTYP = 1, PWAITEN = 1, PBKEN = 1, see section *Common memory space timing register 2..4 (FSMC_PMEM2..4) on page 399* for timing configuration).
2. The CPU performs a byte write in the common memory space, with data byte equal to one Flash command byte (for example 0x00 for Samsung NAND Flash devices). The CLE input of the NAND Flash is active during the write strobe (low pulse on NWE), thus the written byte is interpreted as a command by the NAND Flash. Once the command is latched by the NAND Flash device, it does not need to be written for the following page read operations.
3. The CPU can send the start address (STARTAD) for a read operation by writing four bytes (or three for smaller capacity devices), STARTAD[7:0], then STARTAD[16:9], STARTAD[24:17] and finally STARTAD[25] for 64 Mb x 8 bit NAND Flash) in the common memory or attribute space. The ALE input of the NAND Flash device is active during the write strobe (low pulse on NWE), thus the written bytes are interpreted as

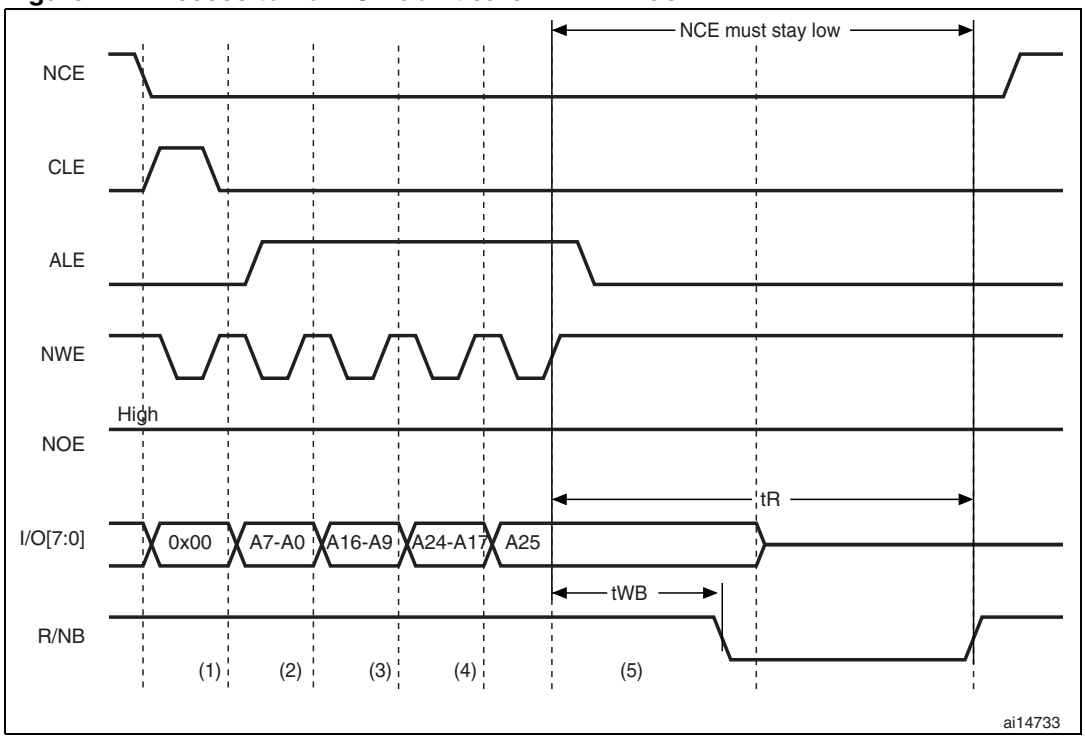
the start address for read operations. Using the attribute memory space makes it possible to use a different timing configuration of the FSMC, which can be used to implement the prewait functionality needed by some NAND Flash memories (see details in *Section 18.6.5: NAND Flash pre-wait functionality on page 395*).

4. The controller waits for the NAND Flash to be ready (R/NB signal high) to become active, before starting a new access (to same or another memory bank). While waiting, the controller maintains the NCE signal active (low).
5. The CPU can then perform byte read operations in the common memory space to read the NAND Flash page (data field + Spare field) byte by byte.
6. The next NAND Flash page can be read without any CPU command or address write operation, in three different ways:
   - by simply performing the operation described in step 5
   - a new random address can be accessed by restarting the operation at step 3
   - a new command can be sent to the NAND Flash device by restarting at step 2

18.6.5 NAND Flash pre-wait functionality

Some NAND Flash devices require that, after writing the last part of the address, the controller wait for the R/NB signal to go low as shown in *Figure 174*.

Figure 174. Access to non 'CE don't care' NAND-Flash



1. CPU wrote byte 0x00 at address 0x7001 0000.
2. CPU wrote byte A7~A0 at address 0x7002 0000.
3. CPU wrote byte A16~A9 at address 0x7002 0000.
4. CPU wrote byte A24~A17 at address 0x7002 0000.
5. CPU wrote byte A25 at address 0x7802 0000: FSMC performs a write access using FSMC_PATT2 timing definition, where ATTHOLD ≥ 7 (providing that (7+1) × HCLK = 112 ns > t_{WB} max). This guarantees that NCE remains low until R/NB goes low and high again (only requested for NAND Flash memories where NCE is not don't care).

When this functionality is needed, it can be guaranteed by programming the MEMHOLD value to meet the t_{WB} timing, however any CPU read or write access to the NAND Flash then has the hold delay of (MEMHOLD + 1) HCLK cycles inserted from the rising edge of the NWE signal to the next access.

To overcome this timing constraint, the attribute memory space can be used by programming its timing register with an ATTHOLD value that meets the t_{WB} timing, and leaving the MEMHOLD value at its minimum. Then, the CPU must use the common memory space for all NAND Flash read and write accesses, except when writing the last address byte to the NAND Flash device, where the CPU must write to the attribute memory space.

18.6.6 Error correction code computation ECC (NAND Flash)

The FSMC PC-Card controller includes two error correction code computation hardware blocks, one per memory bank. They are used to reduce the host CPU workload when processing the error correction code by software in the system.

These two registers are identical and associated with bank 2 and bank 3, respectively. As a consequence, no hardware ECC computation is available for memories connected to bank 4.

The error correction code (ECC) algorithm implemented in the FSMC can perform 1-bit error correction and 2-bit error detection per 256, 512, 1 024, 2 048, 4 096 or 8 192 bytes read from or written to NAND Flash.

The ECC modules monitor the NAND Flash databus and read/write signals (NCE and NWE) each time the NAND Flash memory bank is active.

The functional operations are:

- When access to NAND Flash is made to bank 2 or bank 3, the data present on the D[15:0] bus is latched and used for ECC computation.
- When access to NAND Flash occurs at any other address, the ECC logic is idle, and does not perform any operation. Thus, write operations for defining commands or addresses to NAND Flash are not taken into account for ECC computation.

Once the desired number of bytes has been read from/written to the NAND Flash by the host CPU, the FSMC_ECCR2/3 registers must be read in order to retrieve the computed value. Once read, they should be cleared by resetting the ECCEN bit to zero. To compute a new data block, the ECCEN bit must be set to one in the FSMC_PCR2/3 registers.

18.6.7 NAND Flash/PC Card controller registers

PC Card/NAND Flash control registers 2..4 (FSMC_PCR2..4)

Address offset: 0xA0000000 + 0x40 + 0x20 * (x – 1), x = 2..4

Reset value: 0x0000 0018

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												ECCPS			TAR				TCLR				Res.		ECCEN	PWID		PTYP	PBKEN	PWAITEN	Reserved
												rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	

Bits 19:17 **ECCPS:** ECC page size.
Defines the page size for the extended ECC:
000: 256 bytes
001: 512 bytes
010: 1024 bytes
011: 2048 bytes
100: 4096 bytes
101: 8192 bytes

Bits 16:13 **TAR:** ALE to RE delay.
Sets time from ALE low to RE low in number of AHB clock cycles (HCLK).
Time is: t_ar = (TAR + SET + 4) × THCLK where THCLK is the HCLK clock period
0000: 1 HCLK cycle (default)
1111: 16 HCLK cycles
Note: SET is MEMSET or ATTSET according to the addressed space.

Bits 12:9 **TCLR:** CLE to RE delay.
Sets time from CLE low to RE low in number of AHB clock cycles (HCLK).
Time is t_clr = (TCLR + SET + 4) × THCLK where THCLK is the HCLK clock period
0000: 1 HCLK cycle (default)
1111: 16 HCLK cycles
Note: SET is MEMSET or ATTSET according to the addressed space.

Bits 8:7 Reserved.

Bits 6 **ECCEN:** ECC computation logic enable bit
0: ECC logic is disabled and reset (default after reset),
1: ECC logic is enabled.

Bits 5:4 **PWID:** Databus width.
Defines the external memory device width.
00: 8 bits (default after reset)
01: 16 bits (mandatory for PC Card)
10: reserved, do not use
11: reserved, do not use

Bit 3 **PTYP:** Memory type.
Defines the type of device attached to the corresponding memory bank:
0: PC Card, CompactFlash, CF+ or PCMCIA
1: NAND Flash (default after reset)

Bit 2 **PBKEN:** PC Card/NAND Flash memory bank enable bit.
Enables the memory bank. Accessing a disabled memory bank causes an ERROR on AHB bus
0: Corresponding memory bank is disabled (default after reset),
1: Corresponding memory bank is enabled

Bit 1 **PWAITEN:** Wait feature enable bit.
Enables the Wait feature for the PC Card/NAND Flash memory bank:
0: disabled
1: enabled

Note: For a PC Card, when the wait feature is enabled, the MEMWAITx/ATTWAITx/IOWAITx bits must be programmed to a value higher than $t_{v(IORDY-NOE)}/T_{HCLK} + 4$*, where* $t_{v(IORDY-NOE)}$ *is the maximum time taken by NWAIT to go low once NOE is low.*

Bit 0 Reserved.

FIFO status and interrupt register 2..4 (FSMC_SR2..4)

Address offset: 0xA000 0000 + 0x44 + 0x20 * (x-1), x = 2..4

Reset value: 0x0000 0040

This register contains information about FIFO status and interrupt. The FSMC has a FIFO that is used when writing to memories to store up to16 words of data from the AHB.
This is used to quickly write to the AHB and free it for transactions to peripherals other than the FSMC, while the FSMC is draining its FIFO into the memory. This register has one of its bits that indicates the status of the FIFO, for ECC purposes.
The ECC is calculated while the data are written to the memory, so in order to read the correct ECC the software must wait until the FIFO is empty.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																									FEMPT	IFEN	ILEN	IREN	IFS	ILS	IRS
																									r	rw	rw	rw	rw	rw	rw

Bit 6 **FEMPT:** FIFO empty.
Read-only bit that provides the status of the FIFO
0: FIFO not empty
1: FIFO empty

Bit 5 **IFEN:** Interrupt falling edge detection enable bit
0: Interrupt falling edge detection request disabled.
1: Interrupt falling edge detection request enabled.

Bit 4 **ILEN:** Interrupt high-level detection enable bit
0: Interrupt high-level detection request disabled.
1: Interrupt high-level detection request enabled.

Bit 3 **IREN:** Interrupt rising edge detection enable bit
0: Interrupt rising edge detection request disabled.
1: Interrupt rising edge detection request enabled.

Bit 2 **IFS:** Interrupt falling edge status
The flag is set by hardware and reset by software.
0: No interrupt falling edge occurred.
1: Interrupt falling edge occurred.

Bit 1 **ILS:** Interrupt high-level status
The flag is set by hardware and reset by software.
0: No Interrupt high-level occurred.
1: Interrupt high-level occurred.

Bit 0 **IRS:** Interrupt rising edge status
The flag is set by hardware and reset by software.
0: No interrupt rising edge occurred.
1: Interrupt rising edge occurred.

Common memory space timing register 2..4 (FSMC_PMEM2..4)

Address offset: Address: 0xA000 0000 + 0x48 + 0x20 * (x – 1), x = 2..4

Reset value: 0xFCFC FCFC

Each FSMC_PMEMx (x = 2..4) read/write register contains the timing information for PC Card or NAND Flash memory bank x, used for access to the common memory space of the 16-bit PC Card/CompactFlash, or to access the NAND Flash for command, address write access and data read/write access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEMHIZx								MEMHOLDx								MEMWAITx								MEMSETx							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **MEMHIZx:** Common memory x data bus HiZ time.

Defines the number of HCLK (+1 only for NAND) clock cycles during which the databus is kept in HiZ after the start of a PC-CARD/NAND Flash write access to common memory space on socket x. Only valid for write transaction:

0000 0000: (0x00) 0 HCLK cycle

1111 1111: (0xFF) 255 HCLK cycles (default value after reset)

Bits 23:16 **MEMHOLDx:** Common memory x hold time.

Defines the number of HCLK (+1) clock cycles to hold address (and data for write access) after the command deassertion (NWE, NOE), for PC-CARD/NAND Flash read or write access to common memory space on socket x:

0000 0000: reserved, do not use this value

0000 0001: 1 HCLK cycle to 255 HCLK cycles (default value after reset)

1111 1111: (0xFF)

Bits 15:8 **MEMWAITx:** Common memory x wait time.

Defines the minimum number of HCLK (+1) clock cycles to assert the command (NWE, NOE), for PC-CARD/NAND Flash read or write access to common memory space on socket x. The duration for command assertion is extended if the wait signal (NWAIT) is active (low) at the end of the programmed value of HCLK:

0000 0000: reserved, do not use this value

0000 0001: 2HCLK cycles (+ wait cycle introduced by deasserting NWAIT)

1111 1111: 256 HCLK cycles (+ wait cycle introduced by the Card deasserting NWAIT) (default value after reset)

Bits 7:0 **MEMSETx:** Common memory x setup time.

Defines the number of HCLK (+1 for PC Card, +2 for NAND) clock cycles to set up the address before the command assertion (NWE, NOE), for PC CARD/NAND Flash read or write access to common memory space on socket x:

0000 0000: 1 HCLK cycle

1111 1111: 256 HCLK cycles (default value after reset)

Attribute memory space timing registers 2..4 (FSMC_PATT2..4)

Address offset: 0xA000 0000 + 0x4C + 0x20 * (x – 1), x = 2..4

Reset value: 0xFCFC FCFC

Each FSMC_PATTx (x = 2..4) read/write register contains the timing information for PC Card/CompactFlash or NAND Flash memory bank x. It is used for 8-bit accesses to the attribute memory space of the PC Card/CompactFlash (every AHB transaction is split up into a sequence of 8-bit transactions), or to access the NAND Flash for the last address write access if the timing must differ from that of previous accesses (for Ready/Busy management, refer to *Section 18.6.5: NAND Flash pre-wait functionality*).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ATTHIZx								ATTHOLDx								ATTWAITx								ATTSETx							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **ATTHIZx:** Attribute memory x databus HiZ time.

Defines the number of HCLK clock cycles during which the databus is kept in HiZ after the start of a PC CARD/NAND Flash write access to attribute memory space on socket x. Only valid for write transaction:

0000 0000: 0 HCLK cycle

1111 1111: 255 HCLK cycles (default value after reset)

Bits 23:16 **ATTHOLDx:** Attribute memory x hold time.

Defines the number of HCLK (+1) clock cycles to hold address (and data for write access) after the command deassertion (NWE, NOE), for PC CARD/NAND Flash read or write access to attribute memory space on socket x

0000 0000: 1 HCLK cycle

1111 1111: 256 HCLK cycles (default value after reset)

Bits 15:8 **ATTWAITx:** Attribute memory x wait time.

Defines the minimum number of HCLK (+1) clock cycles to assert the command (NWE, NOE), for PC CARD/NAND Flash read or write access to attribute memory space on socket x. The duration for command assertion is extended if the wait signal (NWAIT) is active (low) at the end of the programmed value of HCLK:

0000 0000: 1 HCLK cycle (+ wait cycle introduced by deassertion of NWAIT)

1111 1111: 256 HCLK cycles (+ wait cycle introduced by the card deasserting NWAIT) (default value after reset)

Bits 7:0 **ATTSETx:** Attribute memory x setup time.

Defines the number of HCLK (+1) clock cycles to set up address before the command assertion (NWE, NOE), for PC CARD/NAND Flash read or write access to attribute memory space on socket x:

0000 0000: 1 HCLK cycle

1111 1111: 256 HCLK cycles (default value after reset)

I/O space timing register 4 (FSMC_PIO4)

Address offset: 0xA000 0000 + 0xB0

Reset value: 0xFCFCFCFC

The FSMC_PIO4 read/write registers contain the timing information used to gain access to the I/O space of the 16-bit PC Card/CompactFlash.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IOHIZx								IOHOLDx								IOWAITx								IOSETx							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **IOHIZx:** I/O x databus HiZ time.

Defines the number of HCLK clock cycles during which the databus is kept in HiZ after the start of a PC Card write access to I/O space on socket x. Only valid for write transaction:

0000 0000: 0 HCLK cycle

1111 1111: 255 HCLK cycles (default value after reset)

Bits 23:16 **IOHOLDx:** I/O x hold time.

Defines the number of HCLK (+1) clock cycles to hold address (and data for write access) after the command deassertion (NWE, NOE), for PC Card read or write access to I/O space on socket x:

0000 0000: 1 HCLK cycle

1111 1111: 256 HCLK cycles (default value after reset)

Bits 15:8 **IOWAITx:** I/O x wait time.

Defines the minimum number of HCLK (+1) clock cycles to assert the command (SMNWE, SMNOE), for PC Card read or write access to I/O space on socket x. The duration for command assertion is extended if the wait signal (NWAIT) is active (low) at the end of the programmed value of HCLK:

0000 0000: reserved, do not use this value

0000 0001: 2 HCLK cycles (+ wait cycle introduced by deassertion of NWAIT)

1111 1111: 256 HCLK cycles (+ wait cycle introduced by the Card deasserting NWAIT) (default value after reset)

Bits 7:0 **IOSETx:** I/O x setup time.

Defines the number of HCLK (+1) clock cycles to set up the address before the command assertion (NWE, NOE), for PC Card read or write access to I/O space on socket x:

0000 0000: 1 HCLK cycle

1111 1111: 256 HCLK cycles (default value after reset)

ECC result registers 2/3 (FSMC_ECCR2/3)

Address offset: 0xA000 0000 + 0x54 + 0x20 * (x – 1), x = 2 or 3

Reset value: 0x0000 0000

These registers contain the current error correction code value computed by the ECC computation modules of the FSMC controller (one module per NAND Flash memory bank). When the CPU reads the data from a NAND Flash memory page at the correct address (refer to *Section 18.6.6: Error correction code computation ECC (NAND Flash)*), the data read from or written to the NAND Flash are processed automatically by ECC computation module. At the end of X bytes read (according to the ECCPS field in the FSMC_PCRx registers), the CPU must read the computed ECC value from the FSMC_ECCx registers, and then verify whether these computed parity data are the same as the parity value recorded in the spare area, to determine whether a page is valid, and, to correct it if applicable. The FSMC_ECCRx registers should be cleared after being read by setting the ECCEN bit to zero. For computing a new data block, the ECCEN bit must be set to one.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECCx																															
r																															

Bits 31:0 **ECCx:** ECC result.

This field provides the value computed by the ECC computation logic. *Table 101* hereafter describes the contents of these bit fields.

Table 101. ECC result relevant bits

ECCPS[2:0]	Page size in bytes	ECC bits
000	256	ECC[21:0]
001	512	ECC[23:0]
010	1024	ECC[25:0]
011	2048	ECC[27:0]
100	4096	ECC[29:0]
101	8192	ECC[31:0]

19 SDIO interface (SDIO)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to high-density performance line devices only.

19.1 SDIO main features

The SD/SDIO MMC card host interface (SDIO) provides an interface between the AHB peripheral bus and MultiMediaCards (MMCs), SD memory cards, SDIO cards and CE-ATA devices.

The MultiMediaCard system specifications are available through the MultiMediaCard Association website at *www.mmca.org*, published by the MMCA technical committee.

SD memory card and SD I/O card system specifications are available through the SD card Association website at *www.sdcard.org*.

CE-ATA system specifications are available through the CE-ATA workgroup website at *www.ce-ata.org*.

The SDIO features include the following:

- Full compliance with *MultiMediaCard System Specification Version 4.2*. Card support for three different databus modes: 1-bit (default), 4-bit and 8-bit
- Full compatibility with previous versions of MultiMediaCards (forward compatibility)
- Full compliance with *SD Memory Card Specifications Version 2.0*
- Full compliance with *SD I/O Card Specification Version 2.0:* card support for two different databus modes: 1-bit (default) and 4-bit
- Full support of the CE-ATA features (full compliance with *CE-ATA digital protocol Rev1.1*)
- Data transfer up to 48 MHz for the 8 bit mode
- Data and command output enable signals to control external bidirectional drivers.

Note: 1 The SDIO does not have an SPI-compatible communication mode.

2 The SD memory card protocol is a superset of the MultiMediaCard protocol as defined in the MultiMediaCard system specification V2.11. Several commands required for SD memory devices are not supported by either SD I/O-only cards or the I/O portion of combo cards. Some of these commands have no use in SD I/O devices, such as erase commands, and thus are not supported in the SDIO. In addition, several commands are different between SD memory cards and SD I/O cards and thus are not supported in the SDIO. For details refer to SD I/O card Specification Version 1.0. CE-ATA is supported over the MMC electrical interface using a protocol that utilizes the existing MMC access primitives. The interface electrical and signaling definition is as defined in the MMC reference.

The MultiMediaCard/SD bus connects cards to the controller.

The current version of the SDIO supports only one SD/SDIO/MMC4.2 card at any one time and a stack of MMC4.1 or previous.

19.2 SDIO bus topology

Communication over the bus is based on command and data transfers.

The basic transaction on the MultiMediaCard/SD/SD I/O bus is the command/response transaction. These types of bus transaction transfer their information directly within the command or response structure. In addition, some operations have a data token.

Data transfers to/from SD/SDIO memory cards are done in data blocks. Data transfers to/from MMC are done data blocks or streams. Data transfers to/from the CE-ATA Devices are done in data blocks.

Figure 175. SDIO “no response” and “no data” operations

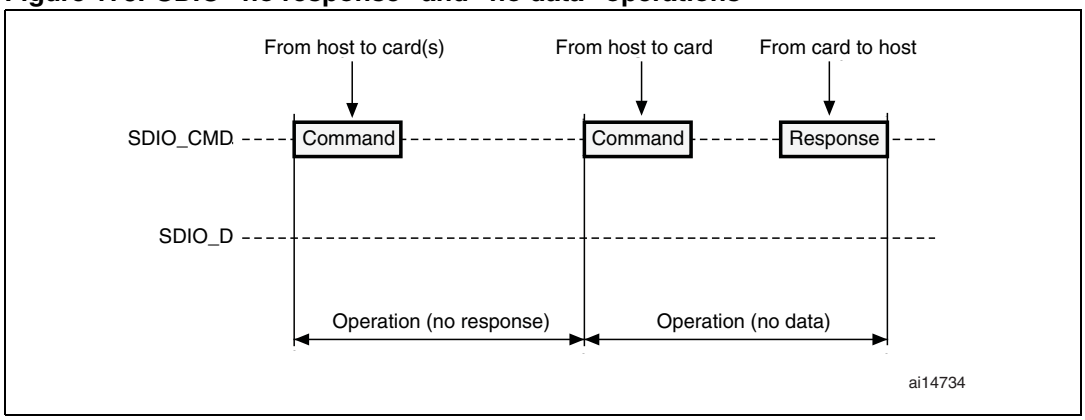


Figure 176. SDIO (multiple) block read operation

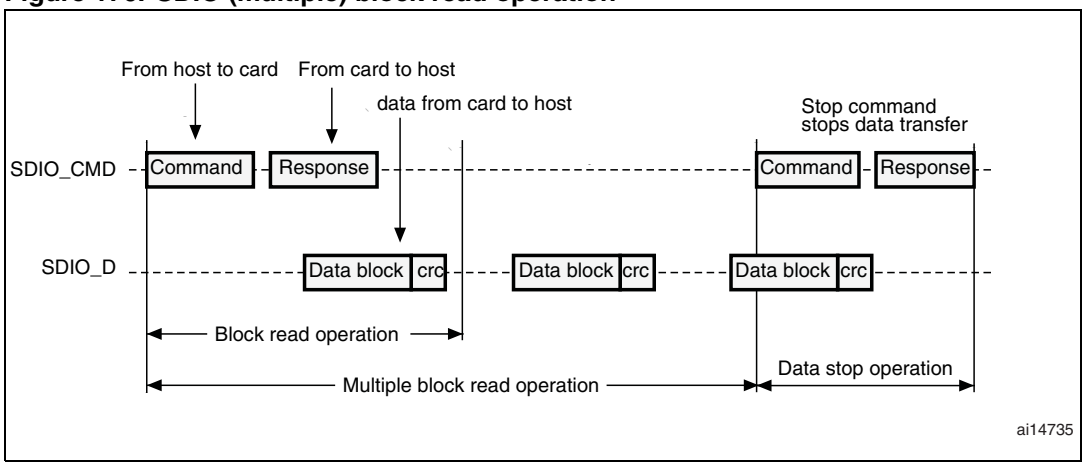
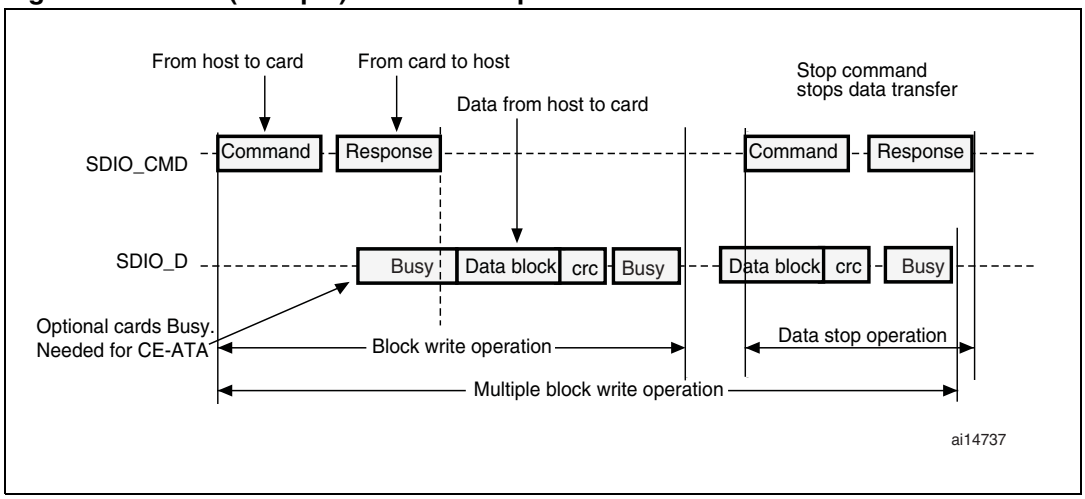


Figure 177. SDIO (multiple) block write operation



Note: The SDIO will not send any data as long as the Busy signal is asserted (SDIO_D0 pulled low).

Figure 178. SDIO sequential read operation

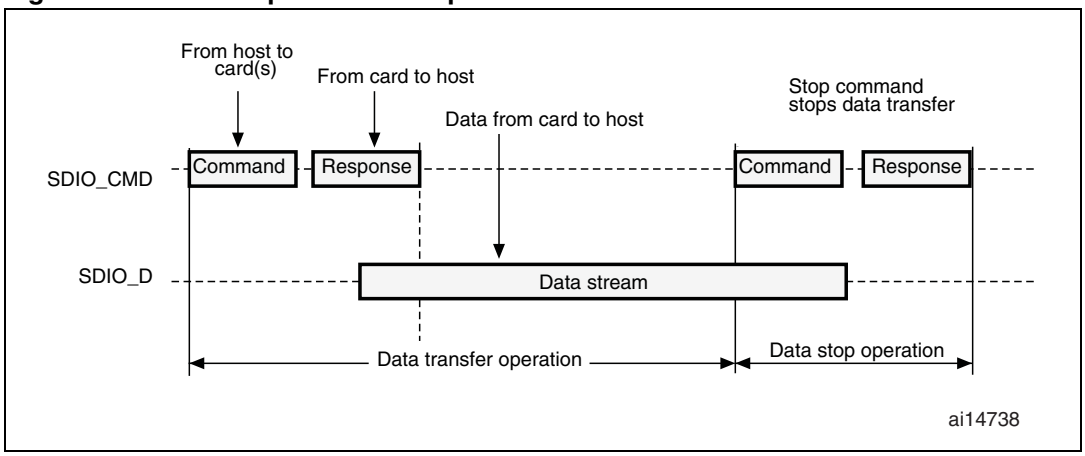
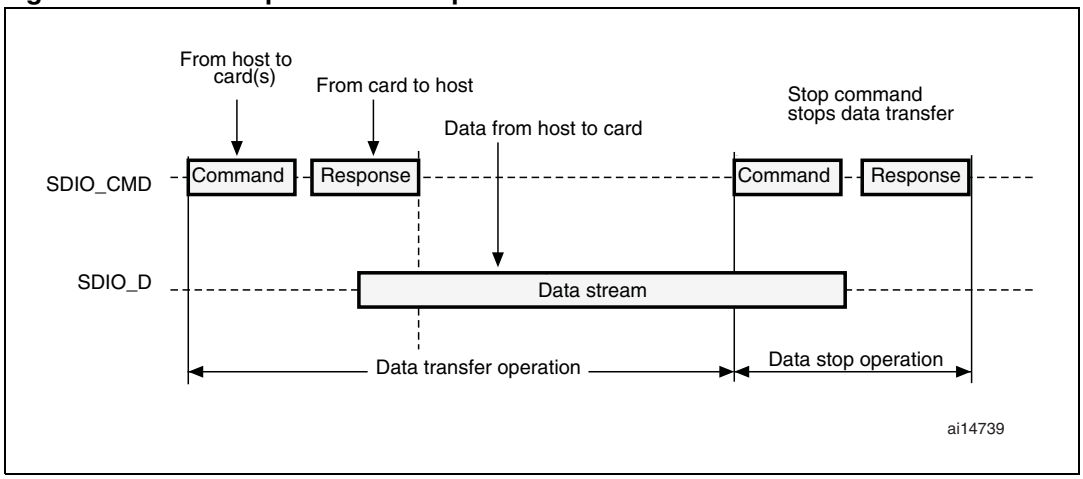


Figure 179. SDIO sequential write operation

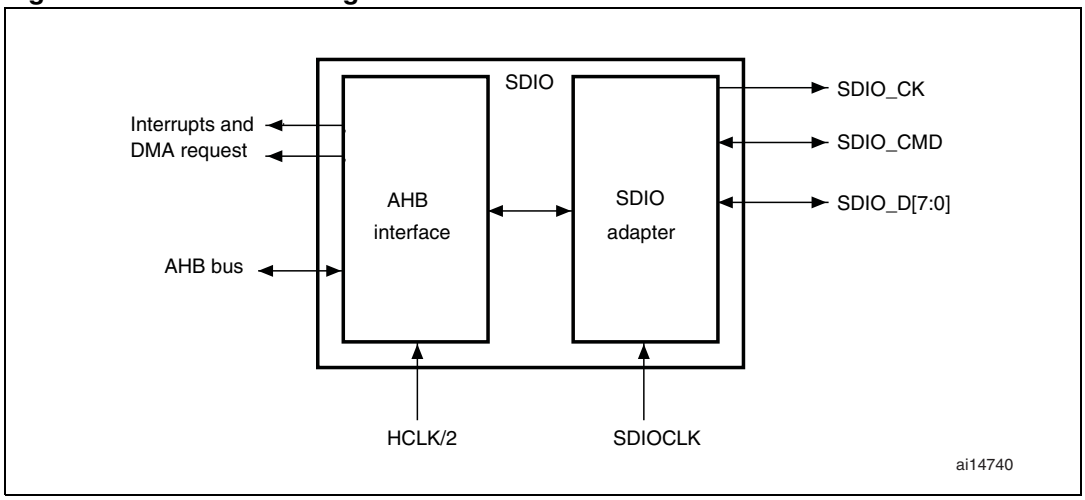


19.3 SDIO functional description

The SDIO consists of two parts:

- The SDIO adapter block provides all functions specific to the MMC/SD/SD I/O card such as the clock generation unit, command and data transfer.
- The AHB interface accesses the SDIO adapter registers, and generates interrupt and DMA request signals.

Figure 180. SDIO block diagram



By default SDIO_D0 is used for data transfer. After initialization, the host can change the databus width.

If a MultiMediaCard is connected to the bus, SDIO_D0, SDIO_D[3:0] or SDIO_D[7:0] can be used for data transfer. MMC V3.31 or previous, supports only 1 bit of data so only SDIO_D0 can be used.

If an SD or SD I/O card is connected to the bus, data transfer can be configured by the host to use SDIO_D0 or SDIO_D[3:0]. All data lines are operating in push-pull mode.

SDIO_CMD has two operational modes:

- Open-drain for initialization (only for MMCV3.31 or previous)
- Push-pull for command transfer (SD/SD I/O card MMC4.2 use push-pull drivers also for initialization)

SDIO_CK is the clock to the card: one bit is transferred on both command and data lines with each clock cycle. The clock frequency can vary between 0 MHz and 20 MHz (for a MultiMediaCard V3.31), between 0 and 48 MHz for a MultiMediaCard V4.0/4.2, or between 0 and 25 MHz (for an SD/SD I/O card).

The SDIO uses two clock signals:

- SDIO adapter clock (SDIOCLK = HCLK)
- AHB bus clock (HCLK/2)

The signals shown in *Table 102* are used on the MultiMediaCard/SD/SD I/O card bus.

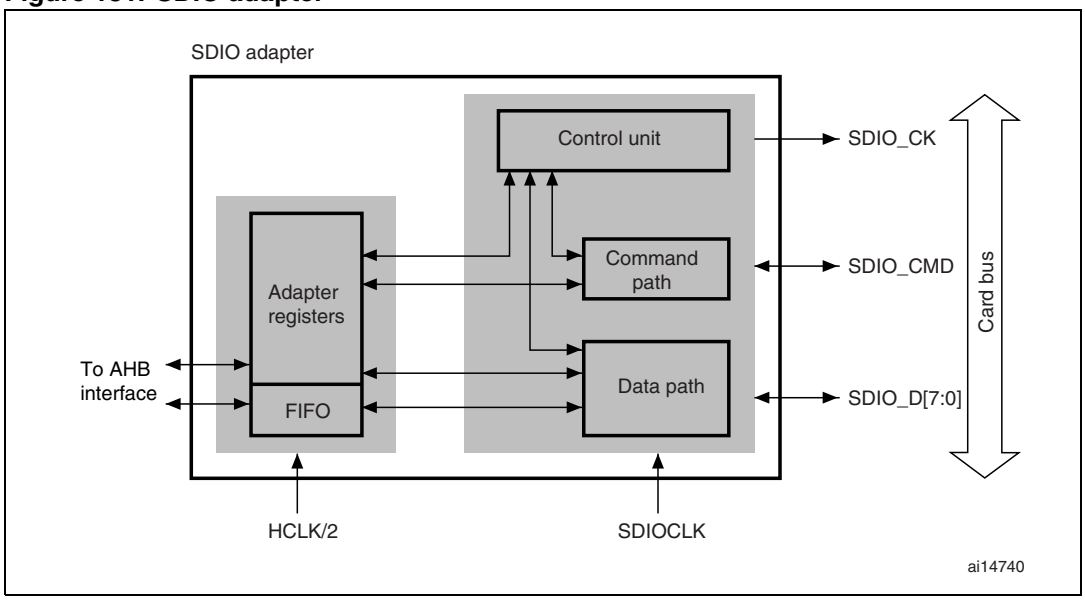
Table 102. SDIO I/O definitions

Pin	Direction	Description
SDIO_CK	Output	MultiMediaCard/SD/SDIO card clock. This pin is the clock from host to card.
SDIO_CMD	Bidirectional	MultiMediaCard/SD/SDIO card command. This pin is the bidirectional command/response signal.
SDIO_D[7:0]	Bidirectional	MultiMediaCard/SD/SDIO card data. These pins are the bidirectional databus.

19.3.1 SDIO adapter

Figure 181 shows a simplified block diagram of an SDIO adapter.

Figure 181. SDIO adapter



The SDIO adapter is a multimedia/secure digital memory card bus master that provides an interface to a multimedia card stack or to a secure digital memory card. It consists of five subunits:

- Adapter register block
- Control unit
- Command path
- Data path
- Data FIFO

Note: The adapter registers and FIFO use the AHB bus clock domain (HCLK/2). The control unit, command path and data path use the SDIO adapter clock domain (SDIOCLK).

Adapter register block

The adapter register block contains all system registers. This block also generates the signals that clear the static flags in the multimedia card. The clear signals are generated when 1 is written into the corresponding bit location in the SDIO Clear register.

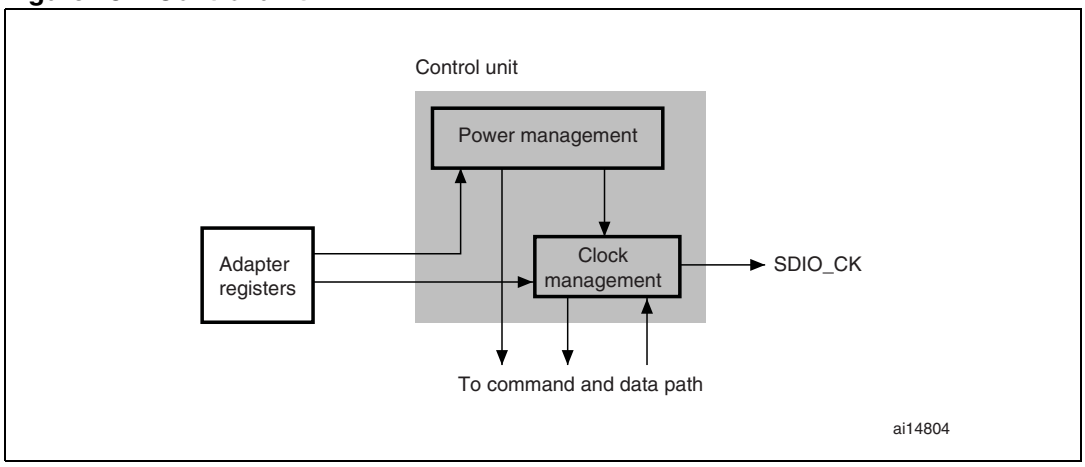
Control unit

The control unit contains the power management functions and the clock divider for the memory card clock.

There are three power phases:

- power-off
- power-up
- power-on

Figure 182. Control unit



The control unit is illustrated in *Figure 182*. It consists of a power management subunit and a clock management subunit.

The power management subunit disables the card bus output signals during the power-off and power-up phases.

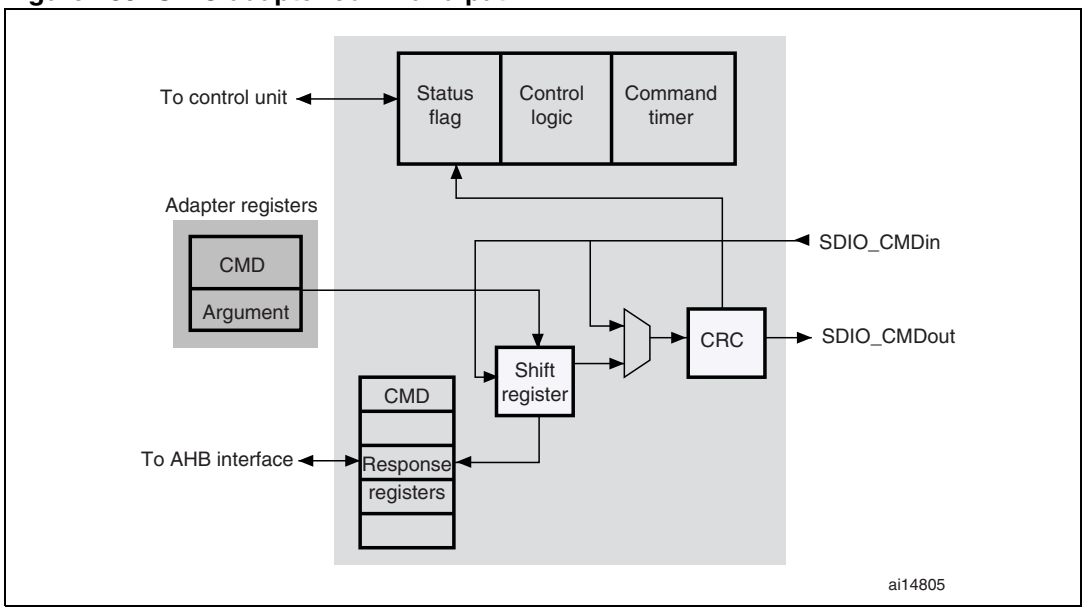
The clock management subunit generates and controls the SDIO_CK signal. The SDIO_CK output can use either the clock divide or the clock bypass mode. The clock output is inactive:

- after reset
- during the power-off or power-up phases
- if the power saving mode is enabled and the card bus is in the Idle state (eight clock periods after both the command and data path subunits enter the Idle phase)

Command path

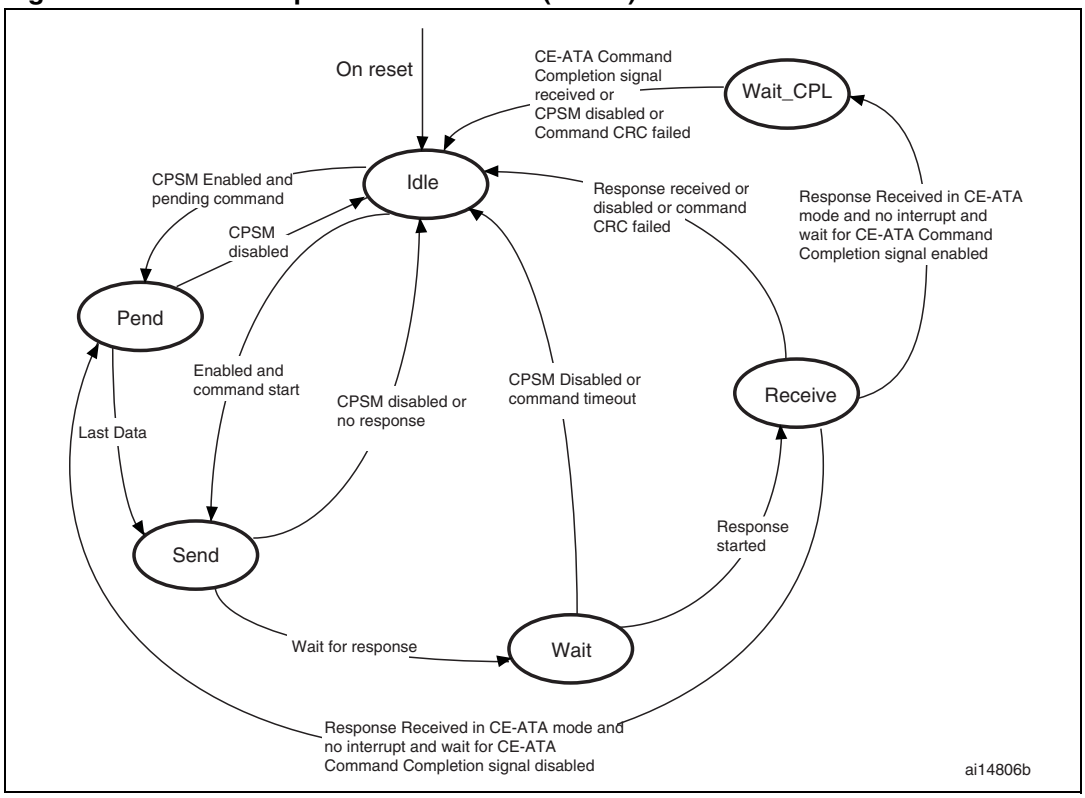
The command path unit sends commands to and receives responses from the cards.

Figure 183. SDIO adapter command path



- Command path state machine (CPSM)
  - When the command register is written to and the enable bit is set, command transfer starts. When the command has been sent, the command path state machine (CPSM) sets the status flags and enters the Idle state if a response is not required. If a response is required, it waits for the response (see *Figure 184 on page 410*). When the response is received, the received CRC code and the internally generated code are compared, and the appropriate status flags are set.

Figure 184. Command path state machine (CPSM)



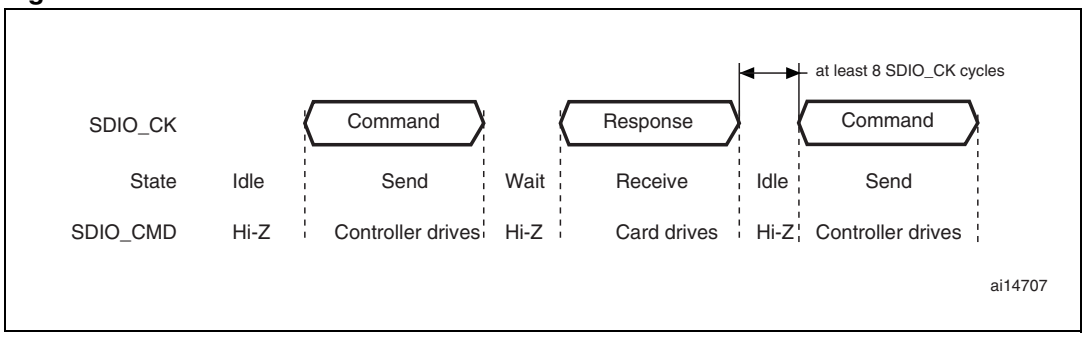
When the Wait state is entered, the command timer starts running. If the timeout is reached before the CPSM moves to the Receive state, the timeout flag is set and the Idle state is entered.

Note: The command timeout has a fixed value of 64 SDIO_CK clock periods.

If the interrupt bit is set in the command register, the timer is disabled and the CPSM waits for an interrupt request from one of the cards. If a pending bit is set in the command register, the CPSM enters the Pend state, and waits for a CmdPend signal from the data path subunit. When CmdPend is detected, the CPSM moves to the Send state. This enables the data counter to trigger the stop command transmission.

Note: The CPSM remains in the Idle state for at least eight SDIO_CK periods to meet the N_{CC} and N_{RC} timing constraints. N_{CC} is the minimum delay between two host commands, and N_{RC} is the minimum delay between the host command and the card response.

Figure 185. SDIO command transfer



- Command format
  - Command: a command is a token that starts an operation. Command are sent from the host either to a single card (addressed command) or to all connected cards (broadcast command are available for MMC V3.31 or previous). Commands are transferred serially on the CMD line. All commands have a fixed length of 48 bits. The general format for a command token for MultiMediaCards, SD-Memory cards and SDIO-Cards is shown in *Table 103*. CE-ATA commands are an extension of MMC commands V4.2, and so have the same format.

    The command path operates in a half-duplex mode, so that commands and responses can either be sent or received. If the CPSM is not in the Send state, the SDIO_CMD output is in the Hi-Z state, as shown in *Figure 185 on page 411*. Data on SDIO_CMD are synchronous with the rising edge of SDIO_CK. *Table* shows the command format.

Table 103. Command format

Bit position	Width	Value	Description
47	1	0	Start bit
46	1	1	Transmission bit
[45:40]	6	-	Command index
[39:8]	32	-	Argument
[7:1]	7	-	CRC7
0	1	1	End bit

  - Response: a response is a token that is sent from an addressed card (or synchronously from all connected cards for MMC V3.31 or previous), to the host as an answer to a previously received command. Responses are transferred serially on the CMD line.

The SDIO supports two response types. Both use CRC error checking:

- 48 bit short response
- 136 bit long response

Note: If the response does not contain a CRC (CMD1 response), the device driver must ignore the CRC failed status.

Table 104. Short response format

Bit position	Width	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	-	Command index
[39:8]	32	-	Argument
[7:1]	7	-	CRC7(or 1111111)
0	1	1	End bit

Table 105. Long response format

Bit position	Width	Value	Description
135	1	0	Start bit
134	1	0	Transmission bit
[133:128]	6	111111	Reserved
[127:1]	127	-	CID or CSD (including internal CRC7)
0	1	1	End bit

The command register contains the command index (six bits sent to a card) and the command type. These determine whether the command requires a response, and whether the response is 48 or 136 bits long (see *Section 19.9.4 on page 445*). The command path implements the status flags shown in *Table 106*:

Table 106. Command path status flags

Flag	Description
CMDREND	Set if response CRC is OK.
CCRCFAIL	Set if response CRC fails.
CMDSENT	Set when command (that does not require response) is sent
CTIMEOUT	Response timeout.
CMDACT	Command transfer in progress.

The CRC generator calculates the CRC checksum for all bits before the CRC code. This includes the start bit, transmitter bit, command index, and command argument (or card status). The CRC checksum is calculated for the first 120 bits of CID or CSD for the long response format. Note that the start bit, transmitter bit and the six reserved bits are not used in the CRC calculation.

The CRC checksum is a 7-bit value:

$$CRC[6:0] = \text{Remainder}\ [(M(x) * x^7) / G(x)]$$

$$G(x) = x^7 + x^3 + 1$$

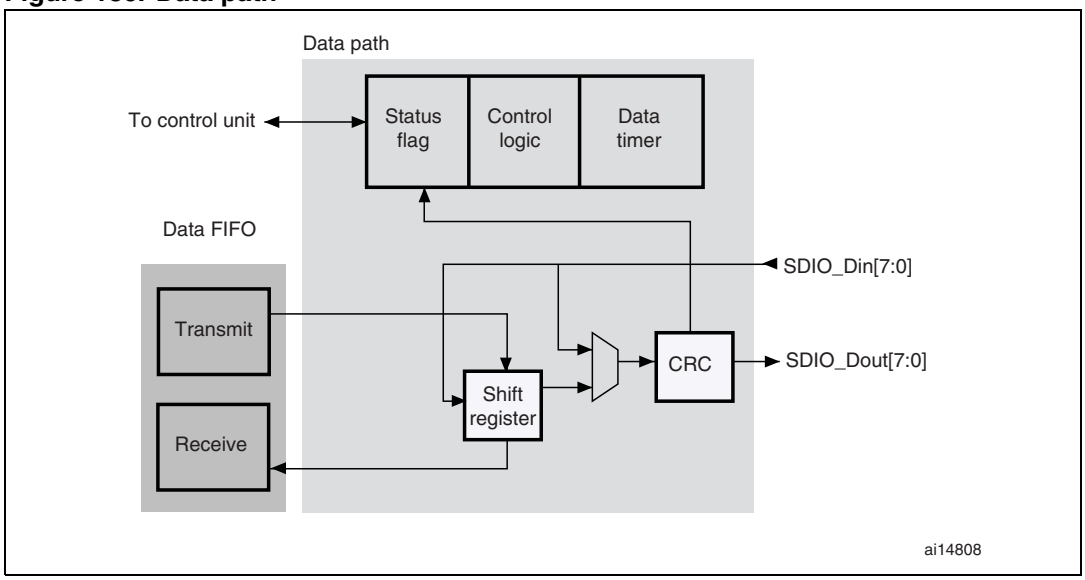
$$M(x) = (\text{start bit}) * x^{39} + ... + (\text{last bit before CRC}) * x^0, \text{ or}$$

$$M(x) = (\text{start bit}) * x^{119} + ... + (\text{last bit before CRC}) * x^0$$

Data path

The data path subunit transfers data to and from cards. *Figure 186* shows a block diagram of the data path.

Figure 186. Data path



The card databus width can be programmed using the clock control register. If the 4-bit wide bus mode is enabled, data is transferred at four bits per clock cycle over all four data signals (SDIO_D[3:0]). If the 8-bit wide bus mode is enabled, data is transferred at eight bits per clock cycle over all eight data signals (SDIO_D[7:0]). If the wide bus mode is not enabled, only one bit per clock cycle is transferred over SDIO_D0.

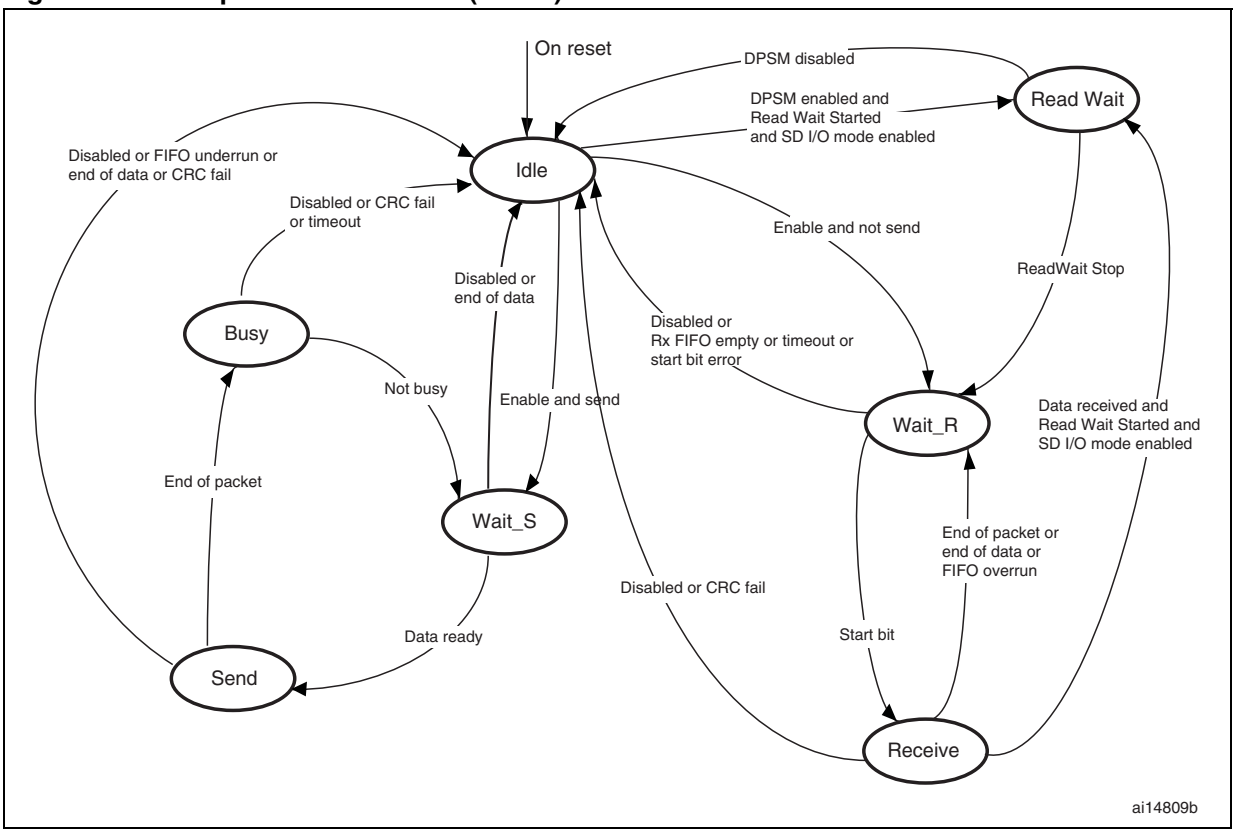
Depending on the transfer direction (send or receive), the data path state machine (DPSM) moves to the Wait_S or Wait_R state when it is enabled:

- Send: the DPSM moves to the Wait_S state. If there is data in the transmit FIFO, the DPSM moves to the Send state, and the data path subunit starts sending data to a card.
- Receive: the DPSM moves to the Wait_R state and waits for a start bit. When it receives a start bit, the DPSM moves to the Receive state, and the data path subunit starts receiving data from a card.

Data path state machine (DPSM)

The DPSM operates at SDIO_CK frequency. Data on the card bus signals is synchronous to the rising edge of SDIO_CK. The DPSM has six states, as shown in *Figure 187: Data path state machine (DPSM)*.

Figure 187. Data path state machine (DPSM)



- Idle: the data path is inactive, and the SDIO_D[7:0] outputs are in Hi-Z. When the data control register is written and the enable bit is set, the DPSM loads the data counter with a new value and, depending on the data direction bit, moves to either the Wait_S or the Wait_R state.
- Wait_R: if the data counter equals zero, the DPSM moves to the Idle state when the receive FIFO is empty. If the data counter is not zero, the DPSM waits for a start bit on SDIO_D. The DPSM moves to the Receive state if it receives a start bit before a timeout, and loads the data block counter. If it reaches a timeout before it detects a start bit, or a start bit error occurs, it moves to the Idle state and sets the timeout status flag.
- Receive: serial data received from a card is packed in bytes and written to the data FIFO. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
  - In block mode, when the data block counter reaches zero, the DPSM waits until it receives the CRC code. If the received code matches the internally generated CRC code, the DPSM moves to the Wait_R state. If not, the CRC fail status flag is set and the DPSM moves to the Idle state.
  - In stream mode, the DPSM receives data while the data counter is not zero. When the counter is zero, the remaining data in the shift register is written to the data FIFO, and the DPSM moves to the Wait_R state.

  If a FIFO overrun error occurs, the DPSM sets the FIFO error flag and moves to the Idle state:
- Wait_S: the DPSM moves to the Idle state if the data counter is zero. If not, it waits until the data FIFO empty flag is deasserted, and moves to the Send state.

Note: The DPSM remains in the Wait_S state for at least two clock periods to meet the N_{WR} timing requirements, where N_{WR} is the number of clock cycles between the reception of the card response and the start of the data transfer from the host.

- Send: the DPSM starts sending data to a card. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
  - In block mode, when the data block counter reaches zero, the DPSM sends an internally generated CRC code and end bit, and moves to the Busy state.
  - In stream mode, the DPSM sends data to a card while the enable bit is high and the data counter is not zero. It then moves to the Idle state.

  If a FIFO underrun error occurs, the DPSM sets the FIFO error flag and moves to the Idle state.
- Busy: the DPSM waits for the CRC status flag:
  - If it does not receive a positive CRC status, it moves to the Idle state and sets the CRC fail status flag.
  - If it receives a positive CRC status, it moves to the Wait_S state if SDIO_D0 is not low (the card is not busy).

  If a timeout occurs while the DPSM is in the Busy state, it sets the data timeout flag and moves to the Idle state.

  The data timer is enabled when the DPSM is in the Wait_R or Busy state, and generates the data timeout error:
  - When transmitting data, the timeout occurs if the DPSM stays in the Busy state for longer than the programmed timeout period
  - When receiving data, the timeout occurs if the end of the data is not true, and if the DPSM stays in the Wait_R state for longer than the programmed timeout period.
- **Data:** data can be transferred from the card to the host or vice versa. Data is transferred via the data lines. They are stored in a FIFO of 32 words, each word is 32 bits wide.

Table 107. Data token format

Description	Start bit	Data	CRC16	End bit
Block Data	0	-	yes	1
Stream Data	0	-	no	1

Data FIFO

The data FIFO (first-in-first-out) subunit is a data buffer with a transmit and receive unit.

The FIFO contains a 32-bit wide, 32-word deep data buffer, and transmit and receive logic. Because the data FIFO operates in the AHB clock domain (HCLK/2), all signals from the subunits in the SDIO clock domain (SDIOCLK) are resynchronized.

Depending on the TXACT and RXACT flags, the FIFO can be disabled, transmit enabled, or receive enabled. TXACT and RXACT are driven by the data path subunit and are mutually exclusive:

- The transmit FIFO refers to the transmit logic and data buffer when TXACT is asserted
- The receive FIFO refers to the receive logic and data buffer when RXACT is asserted

- Transmit FIFO:

  Data can be written to the transmit FIFO through the AHB interface when the SDIO is enabled for transmission.

  The transmit FIFO is accessible via 32 sequential addresses. The transmit FIFO contains a data output register that holds the data word pointed to by the read pointer. When the data path subunit has loaded its shift register, it increments the read pointer and drives new data out.

  If the transmit FIFO is disabled, all status flags are deasserted. The data path subunit asserts TXACT when it transmits data.

Table 108. Transmit FIFO status flags

Flag	Description
TXFIFOF	Set to high when all 32 transmit FIFO words contain valid data.
TXFIFOE	Set to high when the transmit FIFO does not contain valid data.
TXFIFOHE	Set to high when 8 or more transmit FIFO words are empty. This flag can be used as a DMA request.
TXDAVL	Set to high when the transmit FIFO contains valid data. This flag is the inverse of the TXFIFOE flag.
TXUNDERR	Set to high when an underrun error occurs. This flag is cleared by writing to the SDIO Clear register.

- Receive FIFO

  When the data path subunit receives a word of data, it drives the data on the write databus. The write pointer is incremented after the write operation completes. On the read side, the contents of the FIFO word pointed to by the current value of the read pointer is driven onto the read databus. If the receive FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts RXACT when it receives data. *Table 109* lists the receive FIFO status flags. The receive FIFO is accessible via 32 sequential addresses.

Table 109. Receive FIFO status flags

Flag	Description
RXFIFOF	Set to high when all 32 receive FIFO words contain valid data
RXFIFOE	Set to high when the receive FIFO does not contain valid data.
RXFIFOHF	Set to high when 8 or more receive FIFO words contain valid data. This flag can be used as a DMA request.
RXDAVL	Set to high when the receive FIFO is not empty. This flag is the inverse of the RXFIFOE flag.
RXOVERR	Set to high when an overrun error occurs. This flag is cleared by writing to the SDIO Clear register.

19.3.2 SDIO AHB Interface

The AHB interface generates the interrupt and DMA requests, and accesses the SDIO adapter registers and the data FIFO. It consists of a data path, register decoder, and interrupt/DMA logic.

SDIO Interrupts

The interrupt logic generates an interrupt request signal that is asserted when at least one of the selected status flags is high. A mask register is provided to allow selection of the conditions that will generate an interrupt. A status flag generates the interrupt request if a corresponding mask flag is set.

SDIO/DMA Interface: procedure for data transfers between the SDIO and memory

In the example shown, the transfer is from the SDIO host controller to an MMC (512 bytes using CMD24 (WRITE_BLOCK). The SDIO FIFO is filled by data stored in a memory using the DMA controller.

1. Do the card identification process
2. Increase the SDIO_CK frequency
3. Select the card by sending CMD7
4. Configure the DMA2 as follows:
   a) Enable DMA2 controller and clear any pending interrupts
   b) Program DMA2_Channel4 source address register with memory location base address and DMA2_Channel4 destination address register with SDIO_FIFO register address
   c) Program DMA2_Channel4 control register (memory increment, not peripheral increment, peripheral and source width is word size)
   d) Enable DMA2_Channel4
5. Send CMD24 (WRITE_BLOCK) as follows:
   a) Program the SDIO data length register (SDIO data timer register should be already programmed before the card identification process)
   b) Program the SDIO argument register with the address location of the card where data is to be transferred

c) Program the SDIO command register: CmdIndex with 24 (WRITE_BLOCK); WaitResp with ‘1’ (SDIO card host waits for a response); CPSMEN with ‘1’ (SDIO card host enabled to send a command). Other fields are at their reset value.

d) Wait for SDIO_STA[6] = CMDREND interrupt, then program the SDIO data control register: DTEN with ‘1’ (SDIO card host enabled to send data); DTDIR with ‘0’ (from controller to card); DTMODE with ‘0’ (block data transfer); DMAEN with ‘1’ (DMA enabled); DBLOCKSIZE with 0x9 (512 bytes). Other fields are don’t care.

e) Wait for SDIO_STA[10] = DBCKEND

6. Check that no channels are still enabled by polling the DMA Enabled Channel Status register.

19.4 Card functional description

19.4.1 Card identification mode

While in card identification mode the host resets all cards, validates the operation voltage range, identifies cards and sets a relative card address (RCA) for each card on the bus. All data communications in the card identification mode use the command line (CMD) only.

19.4.2 Card reset

The `GO_IDLE_STATE` command (CMD0) is the software reset command and it puts the MultiMediaCard and SD memory in the Idle state. The `IO_RW_DIRECT` command (CMD52) resets the SD I/O card. After power-up or CMD0, all cards output bus drivers are in the high-impedance state and the cards are initialized with a default relative card address (RCA=0x0001) and with a default driver stage register setting (lowest speed, highest driving current capability).

19.4.3 Operating voltage range validation

All cards can communicate with the SDIO card host using any operating voltage within the specification range. The supported minimum and maximum V_{DD} values are defined in the operation conditions register (OCR) on the card.

Cards that store the card identification number (CID) and card specific data (CSD) in the payload memory are able to communicate this information only under data-transfer V_{DD} conditions. When the SDIO card host module and the card have incompatible V_{DD} ranges, the card is not able to complete the identification cycle and cannot send CSD data. For this purpose, the special commands, `SEND_OP_COND` (CMD1), `SD_APP_OP_COND` (ACMD41 for SD Memory), and `IO_SEND_OP_COND` (CMD5 for SD I/O), are designed to provide a mechanism to identify and reject cards that do not match the V_{DD} range desired by the SDIO card host. The SDIO card host sends the required V_{DD} voltage window as the operand of these commands. Cards that cannot perform data transfer in the specified range disconnect from the bus and go to the inactive state.

By using these commands without including the voltage range as the operand, the SDIO card host can query each card and determine the common voltage range before placing out-of-range cards in the inactive state. This query is used when the SDIO card host is able to select a common voltage range or when the user requires notification that cards are not usable.

19.4.4 Card identification process

The card identification process differs for MultiMediaCards and SD cards. For MultiMediaCard cards, the identification process starts at clock rate F_{od}. The SDIO_CMD line output drivers are open-drain and allow parallel card operation during this process. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDIO card host broadcasts `SEND_OP_COND` (CMD1) to receive operation conditions.
3. The response is the wired AND operation of the operation condition registers from all cards.
4. Incompatible cards are placed in the inactive state.
5. The SDIO card host broadcasts `ALL_SEND_CID` (CMD2) to all active cards.
6. The active cards simultaneously send their CID numbers serially. Cards with outgoing CID bits that do not match the bits on the command line stop transmitting and must wait for the next identification cycle. One card successfully transmits a full CID to the SDIO card host and enters the Identification state.
7. The SDIO card host issues `SET_RELATIVE_ADDR` (CMD3) to that card. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state, it does not react to further identification cycles, and its output switches from open-drain to push-pull.
8. The SDIO card host repeats steps 5 through 7 until it receives a timeout condition.

For the SD card, the identification process starts at clock rate F_{od}, and the SDIO_CMD line output drives are push-pull drivers instead of open-drain. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDIO card host broadcasts `SD_APP_OP_COND` (ACMD41).
3. The cards respond with the contents of their operation condition registers.
4. The incompatible cards are placed in the inactive state.
5. The SDIO card host broadcasts `ALL_SEND_CID` (CMD2) to all active cards.
6. The cards send back their unique card identification numbers (CIDs) and enter the Identification state.
7. The SDIO card host issues `SET_RELATIVE_ADDR` (CMD3) to an active card with an address. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state. The SDIO card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.
8. The SDIO card host repeats steps 5 through 7 with all active cards.

For the SD I/O card, the registration process is accomplished as follows:

1. The bus is activated.
2. The SDIO card host sends `IO_SEND_OP_COND` (CMD5).
3. The cards respond with the contents of their operation condition registers.
4. The incompatible cards are set to the inactive state.
5. The SDIO card host issues `SET_RELATIVE_ADDR` (CMD3) to an active card with an address. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state. The

SDIO card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.

19.4.5 Block write

During block write (CMD24 - 27) one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. A card supporting block write is always able to accept a block of data defined by WRITE_BL_LEN. If the CRC fails, the card indicates the failure on the SDIO_D line and the transferred data are discarded and not written, and all further transmitted blocks (in multiple block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block aligned and, block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card will detect the block misalignment error before the beginning of the first misaligned block. (ADDRESS_ERROR error bit is set in the status register). The write operation will also be aborted if the host tries to write over a write-protected area. In this case, however, the card will set the WP_VIOLATION bit.

Programming of the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in ROM, then this unchangeable part must match the corresponding part of the receive buffer. If this match fails, then the card reports an error and does not change any register contents. Some cards may require long and unpredictable times to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing and holds the SDIO_D line low if its write buffer is full and unable to accept new data from a new `WRITE_BLOCK` command. The host may poll the status of the card with a `SEND_STATUS` command (CMD13) at any time, and the card will respond with its status. The READY_FOR_DATA status bit indicates whether the card can accept new data or whether the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card), which will place the card in the Disconnect state and release the SDIO_D line(s) without interrupting the write operation. When reselecting the card, it will reactivate busy indication by pulling SDIO_D to low if programming is still in progress and the write buffer is unavailable.

19.4.6 Block read

In Block read mode the basic unit of data transfer is a block whose maximum size is defined in the CSD (READ_BL_LEN). If READ_BL_PARTIAL is set, smaller blocks whose start and end addresses are entirely contained within one physical block (as defined by READ_BL_LEN) may also be transmitted. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17 (`READ_SINGLE_BLOCK`) initiates a block read and after completing the transfer, the card returns to the Transfer state.

CMD18 (`READ_MULTIPLE_BLOCK`) starts a transfer of several consecutive blocks.

The host can abort reading at any time, within a multiple block operation, regardless of its type. Transaction abort is done by sending the stop transmission command.

If the card detects an error (for example, out of range, address misalignment or internal error) during a multiple block read operation (both types) it stops the data transmission and remains in the data state. The host must than abort the operation by sending the stop transmission command. The read error is reported in the response to the stop transmission command.

If the host sends a stop transmission command after the card transmits the last block of a multiple block operation with a predefined number of blocks, it is responded to as an illegal command, since the card is no longer in the data state. If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed, the card detects a block misalignment error condition at the beginning of the first misaligned block (ADDRESS_ERROR error bit is set in the status register).

19.4.7 Stream access, stream write and stream read (MultiMediaCard only)

In stream mode, data is transferred in bytes and no CRC is appended at the end of each block.

Stream write (MultiMediaCard only)

`WRITE_DAT_UNTIL_STOP` (CMD20) starts the data transfer from the SDIO card host to the card, beginning at the specified address and continuing until the SDIO card host issues a stop command. When partial blocks are allowed (CSD parameter WRITE_BL_PARTIAL is set), the data stream can start and stop at any address within the card address space, otherwise it can only start and stop at block boundaries. Because the amount of data to be transferred is not determined in advance, a CRC cannot be used. When the end of the memory range is reached while sending data and no stop command is sent by the SD card host, any additional transferred data are discarded.

The maximum clock frequency for a stream write operation is given by the following equation fields of the card-specific data register:

$$\text{Maximumspeed} = \text{MIN}\left(\text{TRANSPEED}, \frac{(8 \times 2^{\text{writebllen}})(-\text{NSAC})}{\text{TAAC} \times \text{R2WFACTOR}}\right)$$

- Maximumspeed = maximum write frequency
- TRANSPEED = maximum data transfer rate
- writebllen = maximum write data block length
- NSAC = data read access time 2 in CLK cycles
- TAAC = data read access time 1
- R2WFACTOR = write speed factor

If the host attempts to use a higher frequency, the card may not be able to process the data and stop programming, set the OVERRUN error bit in the status register, and while ignoring all further data transfer, wait (in the receive data state) for a stop command. The write operation is also aborted if the host tries to write over a write-protected area. In this case, however, the card sets the WP_VIOLATION bit.

Stream read (MultiMediaCard only)

`READ_DAT_UNTIL_STOP` (CMD11) controls a stream-oriented data transfer.

This command instructs the card to send its data, starting at a specified address, until the SDIO card host sends `STOP_TRANSMISSION` (CMD12). The stop command has an execution delay due to the serial command transmission and the data transfer stops after the end bit of the stop command. When the end of the memory range is reached while sending data and no stop command is sent by the SDIO card host, any subsequent data sent are considered undefined.

The maximum clock frequency for a stream read operation is given by the following equation and uses fields of the card specific data register.

$$\text{Maximumspeed} = \text{MIN}(\text{TRANSPEED}, \frac{(8 \times 2^{\text{readbllen}})(-\text{NSAC})}{\text{TAAC} \times \text{R2WFACTOR}})$$

- Maximumspeed = maximum read frequency
- TRANSPEED = maximum data transfer rate
- readbllen = maximum read data block length
- writebllen = maximum write data block length
- NSAC = data read access time 2 in CLK cycles
- TAAC = data read access time 1
- R2WFACTOR = write speed factor

If the host attempts to use a higher frequency, the card is not able to sustain data transfer. If this happens, the card sets the UNDERRUN error bit in the status register, aborts the transmission and waits in the data state for a stop command.

19.4.8 Erase: group erase and sector erase

The erasable unit of the MultiMediaCard is the erase group. The erase group is measured in write blocks, which are the basic writable units of the card. The size of the erase group is a card-specific parameter and defined in the CSD.

The host can erase a contiguous range of Erase Groups. Starting the erase process is a three-step sequence.

First the host defines the start address of the range using the `ERASE_GROUP_START` (CMD35) command, next it defines the last address of the range using the `ERASE_GROUP_END` (CMD36) command and, finally, it starts the erase process by issuing the `ERASE` (CMD38) command. The address field in the erase commands is an Erase Group address in byte units. The card ignores all LSBs below the Erase Group size, effectively rounding the address down to the Erase Group boundary.

If an erase command is received out of sequence, the card sets the ERASE_SEQ_ERROR bit in the status register and resets the whole sequence.

If an out-of-sequence (neither of the erase commands, except SEND_STATUS) command received, the card sets the ERASE_RESET status bit in the status register, resets the erase sequence and executes the last command.

If the erase range includes write protected blocks, they are left intact and only nonprotected blocks are erased. The WP_ERASE_SKIP status bit in the status register is set.

The card indicates that an erase is in progress by holding SDIO_D low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

19.4.9 Wide bus selection or deselection

Wide bus (4-bit bus width) operation mode is selected or deselected using `SET_BUS_WIDTH` (ACMD6). The default bus width after power-up or `GO_IDLE_STATE` (CMD0) is 1 bit. `SET_BUS_WIDTH` (ACMD6) is only valid in a transfer state, which means that the bus width can be changed only after a card is selected by `SELECT/DESELECT_CARD` (CMD7).

19.4.10 Protection management

Three write protection methods for the cards are supported in the SDIO card host module:

1. internal card write protection (card responsibility)
2. mechanical write protection switch (SDIO card host module responsibility only)
3. password-protected card lock operation

Internal card write protection

Card data can be protected against write and erase. By setting the permanent or temporary write-protect bits in the CSD, the entire card can be permanently write-protected by the manufacturer or content provider. For cards that support write protection of groups of sectors by setting the `WP_GRP_ENABLE` bit in the CSD, portions of the data can be protected, and the write protection can be changed by the application. The write protection is in units of `WP_GRP_SIZE` sectors as specified in the CSD. The `SET_WRITE_PROT` and `CLR_WRITE_PROT` commands control the protection of the addressed group. The `SEND_WRITE_PROT` command is similar to a single block read command. The card sends a data block containing 32 write protection bits (representing 32 write protect groups starting at the specified address) followed by 16 CRC bits. The address field in the write protect commands is a group address in byte units.

The card ignores all LSBs below the group size.

Mechanical write protect switch

A mechanical sliding tab on the side of the card allows the user to set or clear the write protection on a card. When the sliding tab is positioned with the window open, the card is write-protected, and when the window is closed, the card contents can be changed. A matched switch on the socket side indicates to the SDIO card host module that the card is write-protected. The SDIO card host module is responsible for protecting the card. The position of the write protect switch is unknown to the internal circuitry of the card.

Password protect

The password protection feature enables the SDIO card host module to lock and unlock a card with a password. The password is stored in the 128-bit PWD register and its size is set in the 8-bit PWD_LEN register. These registers are non-volatile so that a power cycle does not erase them. Locked cards respond to and execute certain commands. This means that the SDIO card host module is allowed to reset, initialize, select, and query for status, however it is not allowed to access data on the card. When the password is set (as indicated by a nonzero value of PWD_LEN), the card is locked automatically after power-up. As with the CSD and CID register write commands, the lock/unlock commands are available in the transfer state only. In this state, the command does not include an address argument and the card must be selected before using it. The card lock/unlock commands have the structure and bus transaction types of a regular single-block write command. The transferred data block includes all of the required information for the command (the password setting mode, the PWD itself, and card lock/unlock). The command data block size is defined by the SDIO card host module before it sends the card lock/unlock command, and has the structure shown in *Table 123*.

The bit settings are as follows:

- ERASE: setting it forces an erase operation. All other bits must be zero, and only the command byte is sent
- LOCK_UNLOCK: setting it locks the card. LOCK_UNLOCK can be set simultaneously with SET_PWD, however not with CLR_PWD
- CLR_PWD: setting it clears the password data
- SET_PWD: setting it saves the password data to memory
- PWD_LEN: it defines the length of the password in bytes
- PWD: the password (new or currently used, depending on the command)

The following sections list the command sequences to set/reset a password, lock/unlock the card, and force an erase.

Setting the password

1. Select a card (`SELECT/DESELECT_CARD`, CMD7), if none is already selected.
2. Define the block length (`SET_BLOCKLEN`, CMD16) to send, given by the 8-bit card lock/unlock mode, the 8-bit PWD_LEN, and the number of bytes of the new password. When a password replacement is done, the block size must take into account that both the old and the new passwords are sent with the command.
3. Send `LOCK/UNLOCK` (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (SET_PWD = 1), the length (PWD_LEN), and the password (PWD) itself. When a password replacement is done, the length value (PWD_LEN) includes the length of both passwords, the old and the new one, and the PWD field includes the old password (currently used) followed by the new password.
4. When the password is matched, the new password and its size are saved into the PWD and PWD_LEN fields, respectively. When the old password sent does not correspond (in size and/or content) to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the password is not changed.

The password length field (PWD_LEN) indicates whether a password is currently set. When this field is nonzero, there is a password set and the card locks itself after power-up. It is possible to lock the card immediately in the current power session by setting the LOCK_UNLOCK bit (while setting the password) or sending an additional command for card locking.

Resetting the password

1. Select a card (`SELECT/DESELECT_CARD`, CMD7), if none is already selected.
2. Define the block length (`SET_BLOCKLEN`, CMD16) to send, given by the 8-bit card lock/unlock mode, the 8-bit `PWD_LEN`, and the number of bytes in the currently used password.
3. Send `LOCK/UNLOCK` (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (CLR_PWD = 1), the length (PWD_LEN) and the password (PWD) itself. The LOCK_UNLOCK bit is ignored.
4. When the password is matched, the PWD field is cleared and PWD_LEN is set to 0. When the password sent does not correspond (in size and/or content) to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the password is not changed.

Locking a card

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Define the block length (SET_BLOCKLEN, CMD16) to send, given by the 8-bit card lock/unlock mode (byte 0 in *Table 123*), the 8-bit PWD_LEN, and the number of bytes of the current password.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (LOCK_UNLOCK = 1), the length (PWD_LEN), and the password (PWD) itself.
4. When the password is matched, the card is locked and the CARD_IS_LOCKED status bit is set in the card status register. When the password sent does not correspond (in size and/or content) to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the lock fails.

It is possible to set the password and to lock the card in the same sequence. In this case, the SDIO card host module performs all the required steps for setting the password (see *Setting the password on page 424*), however it is necessary to set the LOCK_UNLOCK bit in Step 3 when the new password command is sent.

When the password is previously set (PWD_LEN is not 0), the card is locked automatically after power on reset. An attempt to lock a locked card or to lock a card that does not have a password fails and the LOCK_UNLOCK_FAILED error bit is set in the card status register.

Unlocking the card

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Define the block length (SET_BLOCKLEN, CMD16) to send, given by the 8-bit cardlock/unlock mode (byte 0 in *Table 123*), the 8-bit PWD_LEN, and the number of bytes of the current password.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (LOCK_UNLOCK = 0), the length (PWD_LEN), and the password (PWD) itself.
4. When the password is matched, the card is unlocked and the CARD_IS_LOCKED status bit is cleared in the card status register. When the password sent is not correct in size and/or content and does not correspond to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the card remains locked.

The unlocking function is only valid for the current power session. When the PWD field is not clear, the card is locked automatically on the next power-up.

An attempt to unlock an unlocked card fails and the LOCK_UNLOCK_FAILED error bit is set in the card status register.

Forcing erase

If the user has forgotten the password (PWD content), it is possible to access the card after clearing all the data on the card. This forced erase operation erases all card data and all password data.

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Set the block length (SET_BLOCKLEN, CMD16) to 1 byte. Only the 8-bit card lock/unlock byte (byte 0 in *Table 123*) is sent.

3. Send `LOCK/UNLOCK` (CMD42) with the appropriate data byte on the data line including the 16-bit CRC. The data block indicates the mode (ERASE = 1). All other bits must be zero.
4. When the ERASE bit is the only bit set in the data field, all card contents are erased, including the PWD and PWD_LEN fields, and the card is no longer locked. When any other bits are set, the LOCK_UNLOCK_FAILED error bit is set in the card status register and the card retains all of its data, and remains locked.

An attempt to use a force erase on an unlocked card fails and the LOCK_UNLOCK_FAILED error bit is set in the card status register.

19.4.11 Card status register

The response format R1 contains a 32-bit field named card status. This field is intended to transmit the card status information (which may be stored in a local status register) to the host. If not specified otherwise, the status entries are always related to the previously issued command.

Table 110 defines the different entries of the status. The type and clear condition fields in the table are abbreviated as follows:

Type:

- E: error bit
- S: status bit
- R: detected and set for the actual command response
- X: detected and set during command execution. The SDIO card host must poll the card by issuing the status command to read these bits.

Clear condition:

- A: according to the card current state
- B: always related to the previous command. Reception of a valid command clears it (with a delay of one command)
- C: clear by read

Table 110. Card status

Bits	Identifier	Type	Value	Description	Clear condition
31	ADDRESS_ OUT_OF_RANGE	E R X	’0’= no error ’1’= error	The command address argument was out of the allowed range for this card. A multiple block or stream read/write operation is (although started in a valid address) attempting to read or write beyond the card capacity.	C
30	ADDRESS_MISALIGN		’0’= no error ’1’= error	The commands address argument (in accordance with the currently set block length) positions the first data block misaligned to the card physical blocks. A multiple block read/write operation (although started with a valid address/block-length combination) is attempting to read or write a data block which is not aligned with the physical blocks of the card.	C
29	BLOCK_LEN_ERROR		’0’= no error ’1’= error	Either the argument of a SET_BLOCKLEN command exceeds the maximum value allowed for the card, or the previously defined block length is illegal for the current command (e.g. the host issues a write command, the current block length is smaller than the maximum allowed value for the card and it is not allowed to write partial blocks)	C
28	ERASE_SEQ_ERROR		’0’= no error ’1’= error	An error in the sequence of erase commands occurred.	C
27	ERASE_PARAM	E X	’0’= no error ’1’= error	An invalid selection of erase groups for erase occurred.	C
26	WP_VIOLATION	E X	’0’= no error ’1’= error	Attempt to program a write-protected block.	C
25	CARD_IS_LOCKED	S R	‘0’ = card unlocked ‘1’ = card locked	When set, signals that the card is locked by the host	A
24	LOCK_UNLOCK_ FAILED	E X	’0’= no error ’1’= error	Set when a sequence or password error has been detected in lock/unlock card command	C
23	COM_CRC_ERROR	E R	’0’= no error ’1’= error	The CRC check of the previous command failed.	B
22	ILLEGAL_COMMAND	E R	’0’= no error ’1’= error	Command not legal for the card state	B
21	CARD_ECC_FAILED	E X	’0’= success ’1’= failure	Card internal ECC was applied but failed to correct the data.	C
20	CC_ERROR	E R	’0’= no error ’1’= error	(Undefined by the standard) A card error occurred, which is not related to the host command.	C

Table 110. Card status (continued)

Bits	Identifier	Type	Value	Description	Clear condition
19	ERROR	E X	’0’= no error ’1’= error	(Undefined by the standard) A generic card error related to the (and detected during) execution of the last host command (e.g. read or write failures).	C
18	Reserved				
17	Reserved				
16	CID/CSD_OVERWRITE	E X	’0’= no error ‘1’= error	Can be either of the following errors: – The CID register has already been written and cannot be overwritten – The read-only section of the CSD does not match the card contents – An attempt to reverse the copy (set as original) or permanent WP (unprotected) bits was made	C
15	WP_ERASE_SKIP	E X	’0’= not protected ’1’= protected	Set when only partial address space was erased due to existing write	C
14	CARD_ECC_DISABLED	S X	’0’= enabled ’1’= disabled	The command has been executed without using the internal ECC.	A
13	ERASE_RESET		’0’= cleared ’1’= set	An erase sequence was cleared before executing because an out of erase sequence command was received (commands other than CMD35, CMD36, CMD38 or CMD13)	C
12:9	CURRENT_STATE	S R	0 = Idle 1 = Ready 2 = Ident 3 = Stby 4 = Tran 5 = Data 6 = Rcv 7 = Prg 8 = Dis 9 = Btst 10-15 = reserved	The state of the card when receiving the command. If the command execution causes a state change, it will be visible to the host in the response on the next command. The four bits are interpreted as a binary number between 0 and 15.	B
8	READY_FOR_DATA	S R	’0’= not ready ‘1’ = ready	Corresponds to buffer empty signalling on the bus	
7	SWITCH_ERROR	E X	’0’= no error ’1’= switch error	If set, the card did not switch to the expected mode as requested by the SWITCH command	B
6	Reserved				
5	APP_CMD	S R	‘0’ = Disabled ‘1’ = Enabled	The card will expect ACMD, or an indication that the command has been interpreted as ACMD	C
4	Reserved for SD I/O Card				

Table 110. Card status (continued)

Bits	Identifier	Type	Value	Description	Clear condition
3	AKE_SEQ_ERROR	E R	’0’= no error ’1’= error	Error in the sequence of the authentication process	C
2	Reserved for application specific commands				
1	Reserved for manufacturer test mode				
0					

19.4.12 SD status register

The SD status contains status bits that are related to the SD memory card proprietary features and may be used for future application-specific usage. The size of the SD Status is one data block of 512 bits. The contents of this register are transmitted to the SDIO card host if ACMD13 is sent (CMD55 followed with CMD13). ACMD13 can be sent to a card in transfer state only (card is selected).

Table 111 defines the different entries of the SD status register. The type and clear condition fields in the table are abbreviated as follows:

Type:

- E: error bit
- S: status bit
- R: detected and set for the actual command response
- X: detected and set during command execution. The SDIO card Host must poll the card by issuing the status command to read these bits

Clear condition:

- A: according to the card current state
- B: always related to the previous command. Reception of a valid command clears it (with a delay of one command)
- C: clear by read

Table 111. SD status

Bits	Identifier	Type	Value	Description	Clear condition
511: 510	DAT_BUS_WIDTH	S R	’00’= 1 (default) ‘01’= reserved ‘10’= 4 bit width ‘11’= reserved	Shows the currently defined databus width that was defined by SET_BUS_WIDTH command	A
509	SECURED_MODE	S R	’0’= Not in the mode ’1’= In Secured Mode	Card is in Secured Mode of operation (refer to the “SD Security Specification”).	A
508: 496	Reserved				

Table 111. SD status (continued)

Bits	Identifier	Type	Value	Description	Clear condition
495: 480	SD_CARD_TYPE	S R	'00xxh'= SD Memory Cards as defined in Physical Spec Ver1.01-2.00 ('x'= don't care). The following cards are currently defined: '0000'= Regular SD RD/WR Card. '0001'= SD ROM Card	In the future, the 8 LSBs will be used to define different variations of an SD memory card (each bit will define different SD types). The 8 MSBs will be used to define SD Cards that do not comply with current SD physical layer specification.	A
479: 448	SIZE_OF_PROTE CT ED_AREA	S R	Size of protected area (See below)	(See below)	A
447: 440	SPEED_CLASS	S R	Speed Class of the card (See below)	(See below)	A
439: 432	PERFORMANCE_ MOVE	S R	Performance of move indicated by 1 [MB/s] step. (See below)	(See below)	A
431:428	AU_SIZE	S R	Size of AU (See below)	(See below)	A
427:424	Reserved				
423:408	ERASE_SIZE	S R	Number of AUs to be erased at a time	(See below)	A
407:402	ERASE_TIMEOUT	S R	Timeout value for erasing areas specified by UNIT_OF_ERASE_AU	(See below)	A
401:400	ERASE_OFFSET	S R	Fixed offset value added to erase time.	(See below)	A
399:312	Reserved				
311:0	Reserved for Manufacturer				

SIZE_OF_PROTECTED_AREA

Setting this field differs between standard- and high-capacity cards. In the case of a standard-capacity card, the capacity of protected area is calculated as follows:

Protected area = SIZE_OF_PROTECTED_AREA_* MULT * BLOCK_LEN.

SIZE_OF_PROTECTED_AREA is specified by the unit in MULT*BLOCK_LEN.

In the case of a high-capacity card, the capacity of protected area is specified in this field:

Protected area = SIZE_OF_PROTECTED_AREA

SIZE_OF_PROTECTED_AREA is specified by the unit in bytes.

SPEED_CLASS

This 8-bit field indicates the speed class and the value can be calculated by $P_W/2$ (where P_W is the write performance).

Table 112. Speed class code field

SPEED_CLASS	Value definition
00h	Class 0
01h	Class 2
02h	Class 4
03h	Class 6
04h – FFh	Reserved

PERFORMANCE_MOVE

This 8-bit field indicates Pm (performance move) and the value can be set by 1 [MB/sec] steps. If the card does not move used RUs (recording units), Pm should be considered as infinity. Setting the field to FFh means infinity.

Table 113. Performance move field

PERFORMANCE_MOVE	Value definition
00h	Not defined
01h	1 [MB/sec]
02h	02h 2 [MB/sec]
---------	---------
FEh	254 [MB/sec]
FFh	Infinity

AU_SIZE

This 4-bit field indicates the AU size and the value can be selected in the power of 2 base from 16 KB.

Table 114. AU_SIZE field

AU_SIZE	Value definition
00h	Not defined
01h	16 KB
02h	32 KB
03h	64 KB
04h	128 KB
05h	256 KB
06h	512 KB
07h	1 MB
08h	2 MB
09h	4 MB
Ah – Fh	Reserved

The maximum AU size, which depends on the card capacity, is defined in *Table 115*. The card can be set to any AU size between RU size and maximum AU size.

Table 115. Maximum AU size

Capacity	16 MB-64 MB	128 MB-256 MB	512 MB	1 GB-32 GB
Maximum AU Size	512 KB	1 MB	2 MB	4 MB

ERASE_SIZE

This 16-bit field indicates N_{ERASE}. When N_{ERASE} numbers of AUs are erased, the timeout value is specified by ERASE_TIMEOUT (Refer to *ERASE_TIMEOUT*). The host should determine the proper number of AUs to be erased in one operation so that the host can show the progress of the erase operation. If this field is set to 0, the erase timeout calculation is not supported.

Table 116. Erase size field

ERASE_SIZE	Value definition
0000h	Erase timeout calculation is not supported.
0001h	1 AU
0002h	2 AU
0003h	3 AU
---------	---------
FFFFh	65535 AU

ERASE_TIMEOUT

This 6-bit field indicates T_{ERASE} and the value indicates the erase timeout from offset when multiple AUs are being erased as specified by ERASE_SIZE. The range of ERASE_TIMEOUT can be defined as up to 63 seconds and the card manufacturer can choose any combination of ERASE_SIZE and ERASE_TIMEOUT depending on the implementation. Determining ERASE_TIMEOUT determines the ERASE_SIZE.

Table 117. Erase timeout field

ERASE_TIMEOUT	Value definition
00	Erase timeout calculation is not supported.
01	1 [sec]
02	2 [sec]
03	3 [sec]
---------	---------
63	63 [sec]

ERASE_OFFSET

This 2-bit field indicates T_{OFFSET} and one of four values can be selected. This field is meaningless if the ERASE_SIZE and ERASE_TIMEOUT fields are set to 0.

Table 118. Erase offset field

ERASE_OFFSET	Value definition
0h	0 [sec]
1h	1 [sec]
2h	2 [sec]
3h	3 [sec]

19.4.13 SD I/O mode

SD I/O interrupts

To allow the SD I/O card to interrupt the MultiMediaCard/SD module, an interrupt function is available on a pin on the SD interface. Pin 8, used as SDIO_D1 when operating in the 4-bit SD mode, signals the cards interrupt to the MultiMediaCard/SD module. The use of the interrupt is optional for each card or function within a card. The SD I/O interrupt is level-sensitive, which means that the interrupt line must be held active (low) until it is either recognized and acted upon by the MultiMediaCard/SD module or deasserted due to the end of the interrupt period. After the MultiMediaCard/SD module has serviced the interrupt, the interrupt status bit is cleared via an I/O write to the appropriate bit in the SD I/O card's internal registers. The interrupt output of all SD I/O cards is active low and the MultiMediaCard/SD module provides pull-up resistors on all data lines (SDIO_D[3:0]). The MultiMediaCard/SD module samples the level of pin 8 (SDIO_D/IRQ) into the interrupt detector only during the interrupt period. At all other times, the MultiMediaCard/SD module ignores this value.

The interrupt period is applicable for both memory and I/O operations. The definition of the interrupt period for operations with single blocks is different from the definition for multiple-block data transfers.

SD I/O suspend and resume

Within a multifunction SD I/O or a card with both I/O and memory functions, there are multiple devices (I/O and memory) that share access to the MMC/SD bus. To share access to the MMC/SD module among multiple devices, SD I/O and combo cards optionally implement the concept of suspend/resume. When a card supports suspend/resume, the MMC/SD module can temporarily halt a data transfer operation to one function or memory (suspend) to free the bus for a higher-priority transfer to a different function or memory. After this higher-priority transfer is complete, the original transfer is resumed (restarted) where it left off. Support of suspend/resume is optional on a per-card basis. To perform the suspend/resume operation on the MMC/SD bus, the MMC/SD module performs the following steps:

1. Determines the function currently using the SDIO_D [3:0] line(s)
2. Requests the lower-priority or slower transaction to suspend
3. Waits for the transaction suspension to complete
4. Begins the higher-priority transaction

5. Waits for the completion of the higher priority transaction
6. Restores the suspended transaction

SD I/O ReadWait

The optional ReadWait (RW) operation is defined only for the SD 1-bit and 4-bit modes. The ReadWait operation allows the MMC/SD module to signal a card that it is reading multiple registers (IO_RW_EXTENDED, CMD53) to temporarily stall the data transfer while allowing the MMC/SD module to send commands to any function within the SD I/O device. To determine when a card supports the ReadWait protocol, the MMC/SD module must test capability bits in the internal card registers. The timing for ReadWait is based on the interrupt period.

19.4.14 Commands and responses

Application-specific and general commands

The SD card host module system is designed to provide a standard interface for a variety of applications types. In this environment, there is a need for specific customer/application features. To implement these features, two types of generic commands are defined in the standard: application-specific commands (ACMD) and general commands (GEN_CMD).

When the card receives the APP_CMD (CMD55) command, the card expects the next command to be an application-specific command. ACMDs have the same structure as regular MultiMediaCard commands and can have the same CMD number. The card recognizes it as ACMD because it appears after APP_CMD (CMD55). When the command immediately following the APP_CMD (CMD55) is not a defined application-specific command, the standard command is used. For example, when the card has a definition for SD_STATUS (ACMD13), and receives CMD13 immediately following APP_CMD (CMD55), this is interpreted as SD_STATUS (ACMD13). However, when the card receives CMD7 immediately following APP_CMD (CMD55) and the card does not have a definition for ACMD7, this is interpreted as the standard (SELECT/DESELECT_CARD) CMD7.

To use one of the manufacturer-specific ACMDs the SD card Host must perform the following steps:

1. Send APP_CMD (CMD55)
   The card responds to the MultiMediaCard/SD module, indicating that the APP_CMD bit is set and an ACMD is now expected.
2. Send the required ACMD
   The card responds to the MultiMediaCard/SD module, indicating that the APP_CMD bit is set and that the accepted command is interpreted as an ACMD. When a non-ACMD is sent, it is handled by the card as a normal MultiMediaCard command and the APP_CMD bit in the card status register stays clear.

When an invalid command is sent (neither ACMD nor CMD) it is handled as a standard MultiMediaCard illegal command error.

The bus transaction for a GEN_CMD is the same as the single-block read or write commands (WRITE_BLOCK, CMD24 or READ_SINGLE_BLOCK,CMD17). In this case, the argument denotes the direction of the data transfer rather than the address, and the data block has vendor-specific format and meaning.

The card must be selected (in transfer state) before sending GEN_CMD (CMD56). The data block size is defined by SET_BLOCKLEN (CMD16). The response to GEN_CMD (CMD56) is in R1b format.

Command types

Both application-specific and general commands are divided into the four following types:

- **broadcast command (BC)**: sent to all cards; no responses returned.
- **broadcast command with response (BCR):** sent to all cards; responses received from all cards simultaneously.
- **addressed (point-to-point) command (AC):** sent to the card that is selected; does not include a data transfer on the SDIO_D line(s).
- **addressed (point-to-point) data transfer command (ADTC):** sent to the card that is selected; includes a data transfer on the SDIO_D line(s).

Command formats

See *Table 103 on page 411* for command formats.

Commands for the MultiMediaCard/SD module

Table 119. Block-oriented write commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD23	ac	[31:16] set to 0 [15:0] number of blocks	R1	SET_BLOCK_COUNT	Defines the number of blocks which are going to be transferred in the multiple-block read or write command that follows.
CMD24	adtc	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	adtc	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows or the requested number of blocks has been received.
CMD26	adtc	[31:0] stuff bits	R1	PROGRAM_CID	Programming of the card identification register. This command must be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for manufacturer.
CMD27	adtc	[31:0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.

Table 120. Block-oriented write protection commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD28	ac	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE).
CMD29	ac	[31:0] data address	R1b	CLR_WRITE_PROT	If the card provides write protection features, this command clears the write protection bit of the addressed group.
CMD30	adtc	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card provides write protection features, this command asks the card to send the status of the write protection bits.
CMD31	Reserved				

Table 121. Erase commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD32 ... CMD34	Reserved. These command indexes cannot be used in order to maintain backward compatibility with older versions of the MultiMediaCard.				
CMD35	ac	[31:0] data address	R1	ERASE_GROUP_START	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31:0] data address	R1	ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	Reserved. This command index cannot be used in order to maintain backward compatibility with older versions of the MultiMediaCards				
CMD38	ac	[31:0] stuff bits	R1	ERASE	Erases all previously selected write blocks.

Table 122. I/O mode commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD39	ac	[31:16] RCA [15:15] register write flag [14:8] register address [7:0] register data	R4	FAST_IO	Used to write and read 8-bit (register) data fields. The command addresses a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the addressed register. This command accesses application-dependent registers that are not defined in the MultiMediaCard standard.
CMD40	bcr	[31:0] stuff bits	R5	GO_IRQ_STATE	Places the system in the interrupt mode.
CMD41	Reserved				

Table 123. Lock card

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD42	adtc	[31:0] stuff bits	R1b	LOCK_UNLOCK	Sets/resets the password or locks/unlocks the card. The size of the data block is set by the SET_BLOCK_LEN command.
CMD43 ... CMD54	Reserved				

Table 124. Application-specific commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD55	ac	[31:16] RCA [15:0] stuff bits	R1	APP_CMD	Indicates to the card that the next command bits is an application specific command rather than a standard command
CMD56	adtc	[31:1] stuff bits [0]: RD/WR			Used either to transfer a data block to the card or to get a data block from the card for general purpose/application-specific commands. The size of the data block shall be set by the SET_BLOCK_LEN command.
CMD57 ... CMD59	Reserved.				
CMD60 ... CMD63	Reserved for manufacturer.				

19.5 Response formats

All responses are sent via the MCCMD command line SDIO_CMD. The response transmission always starts with the left bit of the bit string corresponding to the response code word. The code length depends on the response type.

A response always starts with a start bit (always 0), followed by the bit indicating the direction of transmission (card = 0). A value denoted by x in the tables below indicates a variable entry. All responses, except for the R3 response type, are protected by a CRC. Every command code word is terminated by the end bit (always 1).

There are five types of responses. Their formats are defined as follows:

19.5.1 R1 (normal response command)

Code length = 48 bits. The 45:40 bits indicate the index of the command to be responded to, this value being interpreted as a binary-coded number (between 0 and 63). The status of the card is coded in 32 bits.

Table 125. R1 response

Bit position	Width (bits	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	X	Command index
[39:8]	32	X	Card status
[7:1]	7	X	CRC7
0	1	1	End bit

19.5.2 R1b

It is identical to R1 with an optional busy signal transmitted on the data line. The card may become busy after receiving these commands based on its state prior to the command reception.

19.5.3 R2 (CID, CSD register)

Code length = 136 bits. The contents of the CID register are sent as a response to the CMD2 and CMD10 commands. The contents of the CSD register are sent as a response to CMD9. Only the bits [127...1] of the CID and CSD are transferred, the reserved bit [0] of these registers is replaced by the end bit of the response. The card indicates that an erase is in progress by holding MCDAT low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

Table 126. R2 response

Bit position	Width (bits	Value	Description
135	1	0	Start bit
134	1	0	Transmission bit
[133:128]	6	‘111111’	Command index
[127:1]	127	X	Card status
0	1	1	End bit

19.5.4 R3 (OCR register)

Code length: 48 bits. The contents of the OCR register are sent as a response to CMD1. The level coding is as follows: restricted voltage windows = low, card busy = low.

Table 127. R3 response

Bit position	Width (bits	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	‘111111’	Reserved
[39:8]	32	X	OCR register
[7:1]	7	‘1111111’	Reserved
0	1	1	End bit

19.5.5 R4 (Fast I/O)

Code length: 48 bits. The argument field contains the RCA of the addressed card, the register address to be read out or written to, and its content.

Table 128. R4 response

Bit position		Width (bits	Value	Description
47		1	0	Start bit
46		1	0	Transmission bit
[45:40]		6	‘111111’	Reserved
[39:8] Argument field	[31:16]	16	X	RCA
	[15:8]	8	X	register address
	[7:0]	8	X	read register contents
[7:1]		7	‘1111111’	CRC7
0		1	1	End bit

19.5.6 R4b

For SD I/O only: an SDIO card receiving the CMD5 will respond with a unique SDIO response R4. The format is:

Table 129. R4b response

Bit position		Width (bits	Value	Description
47		1	0	Start bit
46		1	0	Transmission bit
[45:40]		6	x	Reserved
[39:8] Argument field	39	16	X	Card is ready
	[38:36]	3	X	Number of I/O functions
	35	1	X	Present memory
	[34:32]	3	X	Stuff bits
	[31:8]	24	X	I/O ORC

Table 129. R4b response (continued)

Bit position	Width (bits	Value	Description
[7:1]	7	X	Reserved
0	1	1	End bit

Once an SD I/O card has received a CMD5, the I/O portion of that card is enabled to respond normally to all further commands. This I/O enable of the function within the I/O card will remain set until a reset, power cycle or CMD52 with write to I/O reset is received by the card. Note that an SD memory-only card may respond to a CMD5. The proper response for a memory-only card would be *Present memory* = 1 and *Number of I/O functions* = 0. A memory-only card built to meet the SD Memory Card specification version 1.0 would detect the CMD5 as an illegal command and not respond. The I/O aware host will send CMD5. If the card responds with response R4, the host determines the card's configuration based on the data contained within the R4 response.

19.5.7 R5 (interrupt request)

Only for MultiMediaCard. Code length: 48 bits. If the response is generated by the host, the RCA field in the argument will be 0x0.

Table 130. R5 response

Bit position		Width (bits	Value	Description
47		1	0	Start bit
46		1	0	Transmission bit
[45:40]		6	'111111'	CMD40
[39:8] Argument field	[31:16]	16	X	RCA [31:16] of winning card or of the host
	[15:0]	16	X	Not defined. May be used for IRQ data
[7:1]		7	X	CRC7
0		1	1	End bit

19.5.8 R6

Only for SD I/O. The normal response to CMD3 by a memory device. It is shown in *Table 131*.

Table 131. R6 response

Bit position		Width (bits)	Value	Description
47		1	0	Start bit
46		1	0	Transmission bit
[45:40]		6	'101000'	CMD40
[39:8] Argument field	[31:16]	16	X	RCA [31:16] of winning card or of the host
	[15:0]	16	X	Not defined. May be used for IRQ data

Table 131. R6 response (continued)

Bit position	Width (bits)	Value	Description
[7:1]	7	X	CRC7
0	1	1	End bit

The card [23:8] status bits are changed when CMD3 is sent to an I/O-only card. In this case, the 16 bits of response are the SD I/O-only values:

- Bit [15] COM_CRC_ERROR
- Bit [14] ILLEGAL_COMMAND
- Bit [13] ERROR
- Bits [12:0] Reserved

19.6 SDIO I/O card-specific operations

The following features are SD I/O-specific operations:

- SDIO read wait operation by SDIO_D2 signalling
- SDIO read wait operation by stopping the clock
- SDIO suspend/resume operation (write and read suspend)
- SDIO interrupts

The SDIO supports these operations only if the SDIO_DCTRL[11] bit is set, except for read suspend that does not need specific hardware implementation.

19.6.1 SDIO I/O read wait operation by SDIO_D2 signalling

It is possible to start the readwait interval before the first block is received: when the data path is enabled (SDIO_DCTRL[0] bit set), the SDIO-specific operation is enabled (SDIO_DCTRL[11] bit set), read wait starts (SDI0_DCTRL[10] =0 and SDI_DCTRL[8] =1) and data direction is from card to SDIO (SDIO_DCTRL[1] = 1), the DPSM directly moves from Idle to Readwait. In Readwait the DPSM drives SDIO_D2 to 0 after 2 SDIO_CK clock cycles. In this state, when you set the RWSTOP bit (SDIO_DCTRL[9]), the DPSM remains in Wait for two more SDIO_CK clock cycles to drive SDIO_D2 to 1 for one clock cycle (in accordance with SDIO specification). The DPSM then starts waiting again until it receives data from the card. The DPSM will not start a readwait interval while receiving a block even if read wait start is set: the readwait interval will start after the CRC is received. The RWSTOP bit has to be cleared to start a new read wait operation. During the readwait interval, the SDIO can detect SDIO interrupts on SDIO_D1.

19.6.2 SDIO read wait operation by stopping SDIO_CK

If the SDIO card does not support the previous read wait method, the SDIO can perform a read wait by stopping SDIO_CK (SDIO_DCTRL is set just like in the method presented in *Section 19.6.1*, but SDIO_DCTRL[10] =1): DSPM stops the clock two SDIO_CK cycles after the end bit of the current received block and starts the clock again after the read wait start bit is set.

As SDIO_CK is stopped, any command can be issued to the card. During a read/wait interval, the SDIO can detect SDIO interrupts on SDIO_D1.

19.6.3 SDIO suspend/resume operation

While sending data to the card, the SDIO can suspend the write operation. the SDIO_CMD[11] bit is set and indicates to the CPSM that the current command is a suspend command. The CPSM analyzes the response and when the ACK is received from the card (suspend accepted), it acknowledges the DPSM that goes Idle after receiving the CRC token of the current block.

The hardware does not save the number of the remaining block to be sent to complete the suspended operation (resume).

The write operation can be suspended by software, just by disabling the DPSM (SDIO_DCTRL[0] =0) when the ACK of the suspend command is received from the card. The DPSM enters then the Idle state.

To suspend a read: the DPSM waits in the Wait_r state as the function to be suspended sends a complete packet just before stopping the data transaction. The application continues reading RxFIFO until the FIF0 is empty, and the DPSM goes Idle automatically.

19.6.4 SDIO interrupts

SDIO interrupts are detected on the SDIO_D1 line once the SDIO_DCTRL[11] bit is set.

19.7 CE-ATA specific operations

The following features are CE-ATA specific operations:

- sending the command completion signal disable to the CE-ATA device
- receiving the command completion signal from the CE-ATA device
- signaling the completion of the CE-ATA command to the CPU, using the status bit and/or interrupt.

The SDIO supports these operations only for the CE-ATA CMD61 command, that is, if SDIO_CMD[14] is set.

19.7.1 Command completion signal disable

Command completion signal disable is sent 8 bit cycles after the reception of a **short** response if the ‘enable CMD completion’ bit, SDIO_CMD[12], is not set and the ‘not interrupt Enable’ bit, SDIO_CMD[13], is set.

The CPSM enters the Pend state, loading the command shift register with the disable sequence “00001” and, the command counter with 43. Eight cycles after, a trigger moves the CPSM to the Send state. When the command counter reaches 48, the CPSM becomes Idle as no response is awaited.

19.7.2 Command completion signal enable

If the ‘enable CMD completion’ bit SDIO_CMD[12] is set and the ‘not interrupt Enable’ bit SDIO_CMD[13] is set, the CPSM waits for the command completion signal in the Waitcpl state.

When ‘0’ is received on the CMD line, the CPSM enters the Idle state. No new command can be sent for 7 bit cycles. Then, for the last 5 cycles (out of the 7) the CMD line is driven to ‘1’ in push-pull mode.

19.7.3 CE-ATA interrupt

The command completion is signaled to the CPU by the status bit SDIO_STA[23]. This static bit can be cleared with the clear bit SDIO_ICR[23].

The SDIO_STA[23] status bit can generate an interrupt on each interrupt line, depending on the mask bit SDIO_MASKx[23].

19.7.4 Aborting CMD61

If the command completion disable signal has not been sent and CMD61 needs to be aborted, the command state machine must be disabled. It then becomes Idle, and the CMD12 command can be sent. No command completion disable signal is sent during the operation.

19.8 HW flow control

The HW flow control functionality is used to avoid FIFO underrun (TX mode) and overrun (RX mode) errors.

The behavior is to stop SDIO_CK and freeze SDIO state machines. The data transfer is stalled while the FIFO is unable to transmit or receive data. Only state machines clocked by SDIOCLK are frozen, the AHB interface is still alive. The FIFO can thus be filled or emptied even if flow control is activated.

To enable HW flow control, the SDIO_CLKCR[14] register bit must be set to 1. After reset Flow Control is disabled.

19.9 SDIO registers

The device communicates to the system via 32-bit-wide control registers accessible via AHB.

19.9.1 SDIO power control register (SDIO_POWER)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																														PWRCTRL	
																														rw	rw

Bits 31:2 Reserved, always read as 0.

[1:0] **PWRCTRL:** Power supply control bits.
These bits are used to define the current functional state of the card clock:
00: Power-off: the clock to card is stopped.
01: Reserved
10: Reserved power-up
11: Power-on: the card is clocked.

Note: After a data write, data cannot be written to this register for seven HCLK clock periods.

19.9.2 SDI clock control register (SDIO_CLKCR)

Address offset: 0x04

Reset value: 0x0000 0000

The SDIO_CLKCR register controls the SDIO_CK output clock.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																	HWFC_EN	NEGEDGE	WID BUS		BYPASS	PWRSAV	CLKEN	CLKDIV							
																	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, always read as 0.

Bit 14 **HWFC_EN:** HW Flow Control enable

0b: HW Flow Control is disabled

1b: HW Flow Control is enabled

When HW Flow Control is enabled, the meaning of the TXFIFOE and RXFIFOF interrupt signals, please see SDIO Status register definition in *Section 19.9.11*.

Bit 13 **NEGEDGE:**SDIO_CK dephasing selection bit

0b: SDIO_CK generated on the rising edge of the master clock SDIOCLK

1b: SDIO_CK generated on the falling edge of the master clock SDIOCLK

Bits 12:11 **WIDBUS:** Wide bus mode enable bit

00: Default bus mode: SDIO_D0 used.

01: 4-wide bus mode: SDIO_D[3:0] used.

10: 8-wide bus mode: SDIO_D[7:0] used

Bit 10 **BYPASS:** Clock divider bypass enable bit

0: Disable bypass: SDIOCLK is divided according to the CLKDIV value before driving the SDIO_CK output signal.

1: Enable bypass: SDIOCLK directly drives the SDIO_CK output signal.

Bit 9 **PWRSAV:** Power saving configuration bit.

For power saving, the SDIO_CK clock output can be disabled when the bus is idle by setting PWRSAV:

0: SDIO_CK clock is always enabled.

1: SDIO_CK is only enabled when the bus is active.

Bit 8 **CLKEN:** Clock enable bit

0: SDIO_CK is disabled.

1: SDIO_CK is enabled.

Bits 7:0 **CLKDIV:** Clock divide factor.

This field defines the divide factor between the input clock (SDIOCLK) and the output clock (SDIO_CK): SDIO_CK frequency = SDIOCLK / [CLKDIV + 2].

Note: 1 While the SD/SDIO card or MultiMediaCard is in identification mode, the SDIO_CK frequency must be less than 400 kHz.

2 The clock frequency can be changed to the maximum card bus frequency when relative card addresses are assigned to all cards.

3 After a data write, data cannot be written to this register for seven HCLK clock periods. SDIO_CK can also be stopped during the read wait interval for SD I/O cards: in this case the SDIO_CLKCR register does not control SDIO_CK.

19.9.3 SDIO argument register (SDIO_ARG)

Address offset: 0x08

Reset value: 0x0000 0000

The SDIO_ARG register contains a 32-bit command argument, which is sent to a card as part of a command message.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMDARG																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **CMDARG:** Command argument.

Command argument sent to a card as part of a command message. If a command contains an argument, it must be loaded into this register before writing a command to the command register.

19.9.4 SDIO command register (SDIO_CMD)

Address offset: 0x0C

Reset value: 0x0000 0000

The SDIO_CMD register contains the command index and command type bits. The command index is sent to a card as part of a command message. The command type bits control the command path state machine (CPSM).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																	CE-ATACMD	nIEN	ENCMDcompl	SDIOSuspend	CPSMEN	WAITPEND	WAITINT	WAITRESP		CMDINDEX					
																	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, always read as 0.

Bit 14 **ATACMD:** CE-ATA command.

If ATACMD is set, the CPSM transfers CMD61

Bit 13 **nIEN:** not Interrupt Enable.

if this bit is 0, interrupts in the CE-ATA device are enabled.

Bit 12 **ENCMDcompl:** Enable CMD completion.

If this bit is set, the command completion signal is enabled.

Bit 11 **SDIOSuspend:** SD I/O suspend command.
If this bit is set, the command to be sent is a suspend command (to be used only with SDIO card).

Bit 10 **CPSMEN:** Command path state machine (CPSM) Enable bit.
If this bit is set, the CPSM is enabled.

Bit 9 **WAITPEND:** CPSM Waits for ends of data transfer (CmdPend internal signal).
If this bit is set, the CPSM waits for the end of data transfer before it starts sending a command.

Bit 8 **WAITINT:** CPSM Waits for Interrupt Request.
If this bit is set, the CPSM disables command timeout and waits for an interrupt request.

Bits 7:6 **WAITRESP: Wait for response bits**
They are used to configure whether the CPSM is to wait for a response, and if yes, which kind of response.
00: No response, expect CMDSENT flag
01: Short response, expect CMDREND or CCRCFAIL flag
10: No response, expect CMDSENT flag
11: Long response, expect CMDREND or CCRCFAIL flag

Bit 5:0 **CMDINDEX:** Command Index.
The command index is sent to the card as part of a command message.

Note: 1 After a data write, data cannot be written to this register for seven HCLK clock periods.

2 MultiMediaCards can send two kinds of response: short responses, 48 bits long, or long responses,136 bits long. SD card and SD I/O card can send only short responses, the argument can vary according to the type of response: the software will distinguish the type of response according to the sent command. CE-ATA devices send only short responses.

19.9.5 SDIO command response register (SDIO_RESPCMD)

Address offset: 0x10

Reset value: 0x0000 0000

The SDIO_RESPCMD register contains the command index field of the last command response received. If the command response transmission does not contain the command index field (long or OCR response), the RESPCMD field is unknown, although it must contain 111111b (the value of the reserved field from the response).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																										RESPCMD					
																										r	r	r	r	r	r

Bits 31:6 Reserved, always read as 0.

Bits 5:0 **RESPCMD:** Response command index. Read-only bit field. Contains the command index of the last command response received.

19.9.6 SDIO response 0..4 register (SDIO_RESPx)

Address offset: (0x14 + (4*x)); x = 0..4

Reset value: 0x0000 0000

The SDIO_RESP0/1/2/3/4 registers contain the status of a card, which is part of the receive response.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CARDSTATUSx																															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **CARDSTATUSx:** see *Table 132*.

The Card Status size is 32 or 127 bits, depending on the response type.

Table 132. Response type and SDIO_RESPx registers

Register	Short response	Long response
SDIO_RESP1	Card Status[31:0]	Card Status [127:96]
SDIO_RESP2	Unused	Card Status [95:64]
SDIO_RESP3	Unused	Card Status [63:32]
SDIO_RESP4	Unused	Card Status [31:1]0b

The most significant bit of the card status is received first. The SDIO_RESP3 register LSB is always 0b.

19.9.7 SDIO data timer register (SDIO_DTIMER)

Address offset: 0x24

Reset value: 0x0000 0000

The SDIO_DTIMER register contains the data timeout period, in card bus clock periods.

A counter loads the value from the SDIO_DTIMER register, and starts decrementing when the data path state machine (DPSM) enters the Wait_R or Busy state. If the timer reaches 0 while the DPSM is in either of these states, the timeout status flag is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATATIME																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DATATIME:** Data timeout period.
Data timeout period expressed in card bus clock periods.

Note: A data transfer must be written to the data timer register and the data length register before being written to the data control register.

19.9.8 SDIO data length register (SDIO_DLEN)

Address offset: 0x28

Reset value: 0x0000 0000

The SDIO_DLEN register contains the number of data bytes to be transferred. The value is loaded into the data counter when data transfer starts.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DATALENGTH																								
							rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:25 Reserved, always read as 0.

Bits 24:0 **DATALENGTH:** Data length value.
Number of data bytes to be transferred.

Note: For a block data transfer, the value in the data length register must be a multiple of the block size (see SDIO_DCTRL). A data transfer must be written to the data timer register and the data length register before being written to the data control register.

19.9.9 SDIO data control register (SDIO_DCTRL)

Address offset: 0x2C

Reset value: 0x0000 0000

The SDIO_DCTRL register control the data path state machine (DPSM).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																				SDIOEN	RWMOD	RWSTOP	RWSTART	DBLOCKSIZE				DMAEN	DTMODE	DTDIR	DTEN
																				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, always read as 0.

Bit 11 **SDIOEN:** SD I/O enable functions.
If this bit is set, the DPSM performs an SD I/O-card-specific operation.

Bit 10 **RWMOD:** Read wait mode
0: Read Wait control stopping SDIO_CK
1: Read Wait control using SDIO_D2

Bit 9 **RWSTOP:** Read wait stop
0: Read wait in progress if RWSTART bit is set
1: Enable for read wait stop if RWSTART bit is set

Bit 8 **RWSTART:** Read wait start.
If this bit is set, read wait operation starts.

Bits 7:4 **DBLOCKSIZE:** Data block size.

Define the data block length when the block data transfer mode is selected:

0000: (0 decimal) lock length = 2^0 = 1 byte
0001: (1 decimal) lock length = 2^1 = 2 bytes
0010: (2 decimal) lock length = 2^2 = 4 bytes
0011: (3 decimal) lock length = 2^3 = 8 bytes
0100: (4 decimal) lock length = 2^4 = 16 bytes
0101: (5 decimal) lock length = 2^5 = 32 bytes
0110: (6 decimal) lock length = 2^6 = 64 bytes
0111: (7 decimal) lock length = 2^7 = 128 bytes
1000: (8 decimal) lock length = 2^8 = 256 bytes
1001: (9 decimal) lock length = 2^9 = 512 bytes
1010: (10 decimal) lock length = 2^{10} = 1024 bytes
1011: (11 decimal) lock length = 2^{11} = 2048 bytes
1100: (12 decimal) lock length = 2^{12} = 4096 bytes
1101: (13 decimal) lock length = 2^{13} = 8192 bytes
1110: (14 decimal) lock length = 2^{14} = 16384 bytes
1111: (15 decimal) reserved

Bit 3 **DMAEN:** DMA enable bit

0: DMA disabled.
1: DMA enabled.

Bit 2 **DTMODE:** Data transfer mode selection

0: Block data transfer.
1: Stream data transfer.

Bit 1 **DTDIR:** Data transfer direction selection

0: From controller to card.
1: From card to controller.

[0] **DTEN:** Data transfer enabled bit.

Data transfer starts if 1b is written to the DTEN bit. Depending on the direction bit, DTDIR, the DPSM moves to the Wait_S, Wait_R state or Readwait if RW Start is set immediately at the beginning of the transfer. It is not necessary to clear the enable bit after the end of a data transfer but the SDIO_DCTRL must be updated to enable a new data transfer

Note: After a data write, data cannot be written to this register for seven HCLK clock periods.

19.9.10 SDIO data counter register (SDIO_DCOUNT)

Address offset: 0x30

Reset value: 0x0000 0000

The SDIO_DCOUNT register loads the value from the data length register (see SDIO_DLEN) when the DPSM moves from the Idle state to the Wait_R or Wait_S state. As data is transferred, the counter decrements the value until it reaches 0. The DPSM then moves to the Idle state and the data status end flag, DATAEND, is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DATACOUNT																								
							r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:25 Reserved, always read as 0.

Bits 24:0 **DATACOUNT:** Data count value.

When this bit is read, the number of remaining data bytes to be transferred is returned. Write has no effect.

Note: This register should be read only when the data transfer is complete.

19.9.11 SDIO status register (SDIO_STA)

Address offset: 0x34

Reset value: 0x0000 0000

The SDIO_STA register is a read-only register. It contains two types of flag:

- Static flags (bits [23:22,10:0]): these bits remain asserted until they are cleared by writing to the SDIO Interrupt Clear register (see SDIO_ICR)
- Dynamic flags (bits [21:11]): these bits change state depending on the state of the underlying logic (for example, FIFO full and empty flags are asserted and deasserted as data while written to the FIFO)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CEATAEND	SDIOIT	RXDAVL	TXDAVL	RXFIFOE	TXFIFOE	RXFIFOF	TXFIFOF	RXFIFOHF	TXFIFOHE	RXACT	TXACT	CMDACT	DBCKEND	STBITERR	DATAEND	CMDSENT	CMDREND	RXOVERR	TXUNDERR	DTIMEOUT	CTIMEOUT	DCRCFAIL	CCRCFAIL
Res.								r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 Reserved, always read as 0.

Bit 23 **CEATAEND:** CE-ATA command completion signal received for CMD61.

Bit 22 **SDIOIT:** SDIO interrupt received.

Bit 21 **RXDAVL:** Data available in receive FIFO.

Bit 20 **TXDAVL:** Data available in transmit FIFO.

Bit 19 **RXFIFOE:** Receive FIFO empty

Bit 18 **TXFIFOE:** Transmit FIFO empty

When HW Flow Control is enabled, TXFIFOE signals becomes activated when the FIFO contains 2 words.

Bit 17 **RXFIFOF:** Receive FIFO full

When HW Flow Control is enabled, RXFIFOF signals becomes activated 2 words before the FIFO is full.

Bit 16 **TXFIFOF:** Transmit FIFO full.

Bit 15 **RXFIFOHF:** Receive FIFO Half Full: there are at least 8 words in the FIFO.

Bit 14 **TXFIFOHE:** Transmit FIFO Half Empty: at least 8 words can be written into the FIFO.

Bit 13 **RXACT:** Data receive in progress

Bit 12 **TXACT:** Data transmit in progress

Bit 11 **CMDACT:** Command transfer in progress

Bit 10 **DBCKEND:** Data block sent/received (CRC check passed)

Bit 9 **STBITERR:** Start bit not detected on all data signals in wide bus mode

Bit 8 **DATAEND:** Data end (data counter, SDIDCOUNT, is zero)

Bit 7 **CMDSENT:** Command sent (no response required)

Bit 6 **CMDREND:** Command response received (CRC check passed)

Bit 5 **RXOVERR:** Received FIFO overrun error

Bit 4 **TXUNDERR:** Transmit FIFO underrun error

Bit 3 **DTIMEOUT:** Data timeout

Bit 2 **CTIMEOUT:** Command response timeout.
The Command TimeOut period has a fixed value of 64 SDIO_CK clock periods.

Bit 1 **DCRCFAIL:** Data block sent/received (CRC check failed)

Bit 0 **CCRCFAIL:** Command response received (CRC check failed)

19.9.12 SDIO interrupt clear register (SDIO_ICR)

Address offset: 0x38

Reset value: 0x0000 0000

The SDIO_ICR register is a write-only register. Writing a bit with 1b clears the corresponding bit in the SDIO_STA Status register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CEATAENDC	SDIOITC	Reserved											DBCKENDC	STBITERRC	DATAENDC	CMDSENTC	CMDRENDC	RXOVERRC	TXUNDERRC	DTIMEOUTC	CTIMEOUTC	DCRCFAILC	CCRCFAILC
								rw	rw												rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, always read as 0.

Bit 23 **CEATAENDC:** CEATAEND flag clear bit
Set by software to clear the CEATAEND flag.
0: CEATAEND not cleared
1: CEATAEND cleared

Bit 22 **SDIOITC:** SDIOIT flag clear bit
Set by software to clear the SDIOIT flag.
0: SDIOIT not cleared
1: SDIOIT cleared

Bits 21:11 Reserved, always read as 0.

Bit 10 **DBCKENDC:** DBCKEND flag clear bit
Set by software to clear the DBCKEND flag.
0: DBCKEND not cleared
1: DBCKEND cleared

Bit 9 **STBITERRC:** STBITERR flag clear bit
Set by software to clear the STBITERR flag.
0: STBITERR not cleared
1: STBITERR cleared

Bit 8 **DATAENDC:** DATAEND flag clear bit
Set by software to clear the DATAEND flag.
0: DATAEND not cleared
1: DATAEND cleared

Bit 7 **CMDSENTC:** CMDSENT flag clear bit
Set by software to clear the CMDSENT flag.
0: CMDSENT not cleared
1: CMDSENT cleared

Bit 6 **CMDRENDC:** CMDREND flag clear bit
Set by software to clear the CMDREND flag.
0: CMDREND not cleared
1: CMDREND cleared

Bit 5 **RXOVERRC:** RXOVERR flag clear bit
Set by software to clear the RXOVERR flag.
0: RXOVERR not cleared
1: RXOVERR cleared

Bit 4 **TXUNDERRC:** TXUNDERR flag clear bit
Set by software to clear TXUNDERR flag.
0: TXUNDERR not cleared
1: TXUNDERR cleared

Bit 3 **DTIMEOUTC:** DTIMEOUT flag clear bit
Set by software to clear the DTIMEOUT flag.
0: DTIMEOUT not cleared
1: DTIMEOUT cleared

Bit 2 **CTIMEOUTC:** CTIMEOUT flag clear bit
Set by software to clear the CTIMEOUT flag.
0: CTIMEOUT not cleared
1: CTIMEOUT cleared

Bit 1 **DCRCFAILC:** DCRCFAIL flag clear bit
Set by software to clear the DCRCFAIL flag.
0: DCRCFAIL not cleared
1: DCRCFAIL cleared

Bit 0 **CCRCFAILC:** CCRCFAIL flag clear bit
Set by software to clear the CCRCFAIL flag.
0: CCRCFAIL not cleared
1: CCRCFAIL cleared

19.9.13 SDIO mask register (SDIO_MASK)

Address offset: 0x3C

Reset value: 0x0000 0000

The interrupt mask register determines which status flags generate an interrupt request by setting the corresponding bit to 1b.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CEATAENDIE	SDIOITIE	RXDAVLIE	TXDAVLIE	RXFIFOEIE	TXFIFOEIE	RXFIFOFIE	TXFIFOFIE	RXFIFOHFIE	TXFIFOHEIE	RXACTIE	TXACTIE	CMDACTIE	DBCKENDIE	STBITERRIE	DATAENDIE	CMDSENTIE	CMDRENDIE	RXOVERRIE	TXUNDERRIE	DTIMEOUTIE	CTIMEOUTIE	DCRCFAILIE	CCRCFAILIE
								rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, always read as 0.

Bit 23 **CEATAENDIE:** CE-ATA command completion signal received Interrupt Enable

Set and cleared by software to enable/disable the interrupt generated when receiving the CE-ATA command completion signal.

0: CE-ATA command completion signal received interrupt disabled

1: CE-ATA command completion signal received interrupt enabled

Bit 22 **SDIOITIE:** SDIO Mode Interrupt Received Interrupt Enable

Set and cleared by software to enable/disable the interrupt generated when receiving the SDIO mode interrupt.

0: SDIO Mode Interrupt Received interrupt disabled

1: SDIO Mode Interrupt Received interrupt enabled

Bit 21 **RXDAVLIE:** Data available in Rx FIFO Interrupt Enable

Set and cleared by software to enable/disable the interrupt generated by the presence of data available in Rx FIFO.

0: Data available in Rx FIFO interrupt disabled

1: Data available in Rx FIFO interrupt enabled

Bit 20 **TXDAVLIE:** Data available in Tx FIFO Interrupt Enable

Set and cleared by software to enable/disable the interrupt generated by the presence of data available in Tx FIFO.

0: Data available in Tx FIFO interrupt disabled

1: Data available in Tx FIFO interrupt enabled

Bit 19 **RXFIFOEIE:** Rx FIFO Empty Interrupt Enable

Set and cleared by software to enable/disable interrupt caused by Rx FIFO empty.

0: Rx FIFO empty interrupt disabled

1: Rx FIFO empty interrupt enabled

Bit 18 **TXFIFOEIE:** Tx FIFO Empty Interrupt Enable

Set and cleared by software to enable/disable interrupt caused by Tx FIFO empty.

0: Tx FIFO empty interrupt disabled

1: Tx FIFO empty interrupt enabled

Bit 17 **RXFIFOFIE:** Rx FIFO Full Interrupt Enable

Set and cleared by software to enable/disable interrupt caused by Rx FIFO full.

0: Rx FIFO full interrupt disabled

1: Rx FIFO full interrupt enabled

Bit 16 **TXFIFOFIE:** Tx FIFO Full Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO full.
0: Tx FIFO full interrupt disabled
1: Tx FIFO full interrupt enabled

Bit 15 **RXFIFOHFIE:** Rx FIFO Half Full Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by Rx FIFO half full.
0: Rx FIFO half full interrupt disabled
1: Rx FIFO half full interrupt enabled

Bit 14 **TXFIFOHEIE:** Tx FIFO Half Empty Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO half empty.
0: Tx FIFO half empty interrupt disabled
1: Tx FIFO half empty interrupt enabled

Bit 13 **RXACTIE:** Data Receive Acting Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by data being received (data receive acting).
0: Data receive acting interrupt disabled
1: Data receive acting interrupt enabled

Bit 12 **TXACTIE:** Data Transmit Acting Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by data being transferred (data transmit acting).
0: Data transmit acting interrupt disabled
1: Data transmit acting interrupt enabled

Bit 11 **CMDACTIE:** Command Acting Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by a command being transferred (command acting).
0: Command acting interrupt disabled
1: Command acting interrupt enabled

Bit 10 **DBCKENDIE:** Data Block End Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by data block end.
0: Data block end interrupt disabled
1: Data block end interrupt enabled

Bit 9 **STBITERRIE:** Start Bit Error Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by start bit error.
0: Start bit error interrupt disabled
1: Start bit error interrupt enabled

Bit 8 **DATAENDIE:** Data End Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by data end.
0: Data end interrupt disabled
1: Data end interrupt enabled

Bit 7 **CMDSENTIE:** Command Sent Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by sending command.
0: Command sent interrupt disabled
1: Command sent interrupt enabled

Bit 6 **CMDRENDIE:** Command Response Received Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by receiving command response.
0: Command response received interrupt disabled
1: command Response Received interrupt enabled

Bit 5 **RXOVERRIE:** Rx FIFO OverRun Error Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by Rx FIFO overrun error.
0: Rx FIFO overrun error interrupt disabled
1: Rx FIFO overrun error interrupt enabled

Bit 4 **TXUNDERRIE:** Tx FIFO UnderRun Error Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO underrun error.
0: Tx FIFO underrun error interrupt disabled
1: Tx FIFO underrun error interrupt enabled

Bit 3 **DTIMEOUTIE:** Data TimeOut Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by data timeout.
0: Data timeout interrupt disabled
1: Data timeout interrupt enabled

Bit 2 **CTIMEOUTIE:** Command TimeOut Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by command timeout.
0: Command timeout interrupt disabled
1: Command timeout interrupt enabled

Bit 1 **DCRCFAILIE:** Data CRC Fail Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by data CRC failure.
0: Data CRC fail interrupt disabled
1: Data CRC fail interrupt enabled

Bit 0 **CCRCFAILIE:** Command CRC Fail Interrupt Enable
Set and cleared by software to enable/disable interrupt caused by command CRC failure.
0: Command CRC fail interrupt disabled
1: Command CRC fail interrupt enabled

19.9.14 SDIO FIFO counter register (SDIO_FIFOCNT)

Address offset: 0x48

Reset value: 0x0000 0000

The SDIO_FIFOCNT register contains the remaining number of words to be written to or read from the FIFO. The FIFO counter loads the value from the data length register (see SDIO_DLEN) when the data transfer enable bit, DTEN, is set in the data control register (SDIO_DCTRL register) and the DPSM is at the Idle state. If the data length is not word-aligned (multiple of 4), the remaining 1 to 3 bytes are regarded as a word.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								FIFOCOUNT																							
								r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 Reserved, always read as 0.

Bits 23:0 **FIFOCOUNT:** Remaining number of words to be written to or read from the FIFO.

19.9.15 SDIO data FIFO register (SDIO_FIFO)

Address offset: 0x80

Reset value: 0x0000 0000

The receive and transmit FIFOs can be read or written as 32-bit wide registers. The FIFOs contain 32 entries on 32 sequential addresses. This allows the CPU to use its load and store multiple operands to read from/write to the FIFO.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIF0Data																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

bits 31:0 **FIFOData:** Receive and transmit FIFO data.

The FIFO data occupies 32 entries of 32-bit words, from address:
SDIO base + 0x080 to SDIO base + 0xFC.

19.9.16 SDIO register map

The following table summarizes the SDIO registers.

Table 133. SDIO register map

Address offset	Name	31	30	29	28	27	26	25	24	23	22	21	23	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	SDIO_POWER	Reserved																														PWRCTRL	
0x04	SDIO_CLKCR	Reserved																	HWFC_EN	NEGEDGE	WIDBUS		BYPASS	PWRSAV	CLKEN	CLKDIV							
0x08	SDIO_ARG	CMDARG																															
0x0C	SDIO_CMD	Reserved																	CE-ATACMD	nIEN	ENCMDcompl	SDIOSuspend	CPSMEN	WAITPEND	WAITINT	WAITRESP		CMDINDEX					
0x10	SDIO_RESPCMD	Reserved																										RESPCMD					
0x14	SDIO_RESP1	CARDSTATUS1																															
0x18	SDIO_RESP2	CARDSTATUS2																															
0x1C	SDIO_RESP3	CARDSTATUS3																															
0x20	SDIO_RESP4	CARDSTATUS4																															
0x24	SDIO_DTIMER	DATATIME																															
0x28	SDIO_DLEN	Reserved							DATALENGTH																								
0x2C	SDIO_DCTRL	Reserved																				SDIOEN	RWMOD	RWSTOP	RWSTART	DBLOCKSIZE			DMAEN	DTMODE	DTDIR	DTEN	
0x30	SDIO_DCOUNT	Reserved							DATACOUNT																								

Table 133. SDIO register map (continued)

Address offset	Name	31	30	29	28	27	26	25	24	23	22	21	23	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	SDIO_STA	Reserved								CEATAEND	SDIOIT	RXDAVL	TXDAVL	RXFIFOE	TXFIFOE	RXFIFOF	TXFIFOF	RXFIFOHF	TXFIFOHE	RXACT	TXACT	CMDACT	DBCKEND	STBITERR	DATAEND	CMDSENT	CMDREND	RXOVERR	TXUNDERR	DTIMEOUT	CTIMEOUT	DCRCFAIL	CCRCFAIL
0x38	SDIO_ICR	Reserved								CEATAENDC	SDIOITC	Reserved											DBCKENDC	STBITERRC	DATAENDC	CMDSENTC	CMDRENDC	RXOVERRC	TXUNDERRC	DTIMEOUTC	CTIMEOUTC	DCRCFAILC	CCRCFAILC
0x3C	SDIO_MASK	Reserved								CEATAENDIE	SDIOITIE	RXDAVLIE	TXDAVLIE	RXFIFOEIE	TXFIFOEIE	RXFIFOFIE	TXFIFOFIE	RXFIFOHFIE	TXFIFOHEIE	RXACTIE	TXACTIE	CMDACTIE	DBCKENDIE	STBITERRIE	DATAENDIE	CMDSENTIE	CMDRENDIE	RXOVERRIE	TXUNDERRIE	DTIMEOUTIE	CTIMEOUTIE	DCRCFAILIE	CCRCFAILIE
0x48	SDIO_FIFOCNT	Reserved								FIFOCOUNT																							
0x80	SDIO_FIFO	FIF0Data																															

Note: Refer to Table 1 on page 35 for the register boundary addresses.

20 USB full speed device interface (USB)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the STM32F103xx performance line and STM32F102xx USB access line families only.

20.1 USB introduction

The USB peripheral implements an interface between a full-speed USB 2.0 bus and the APB1 bus.

USB suspend/resume are supported which allows to stop the device clocks for low-power consumption.

20.2 USB main features

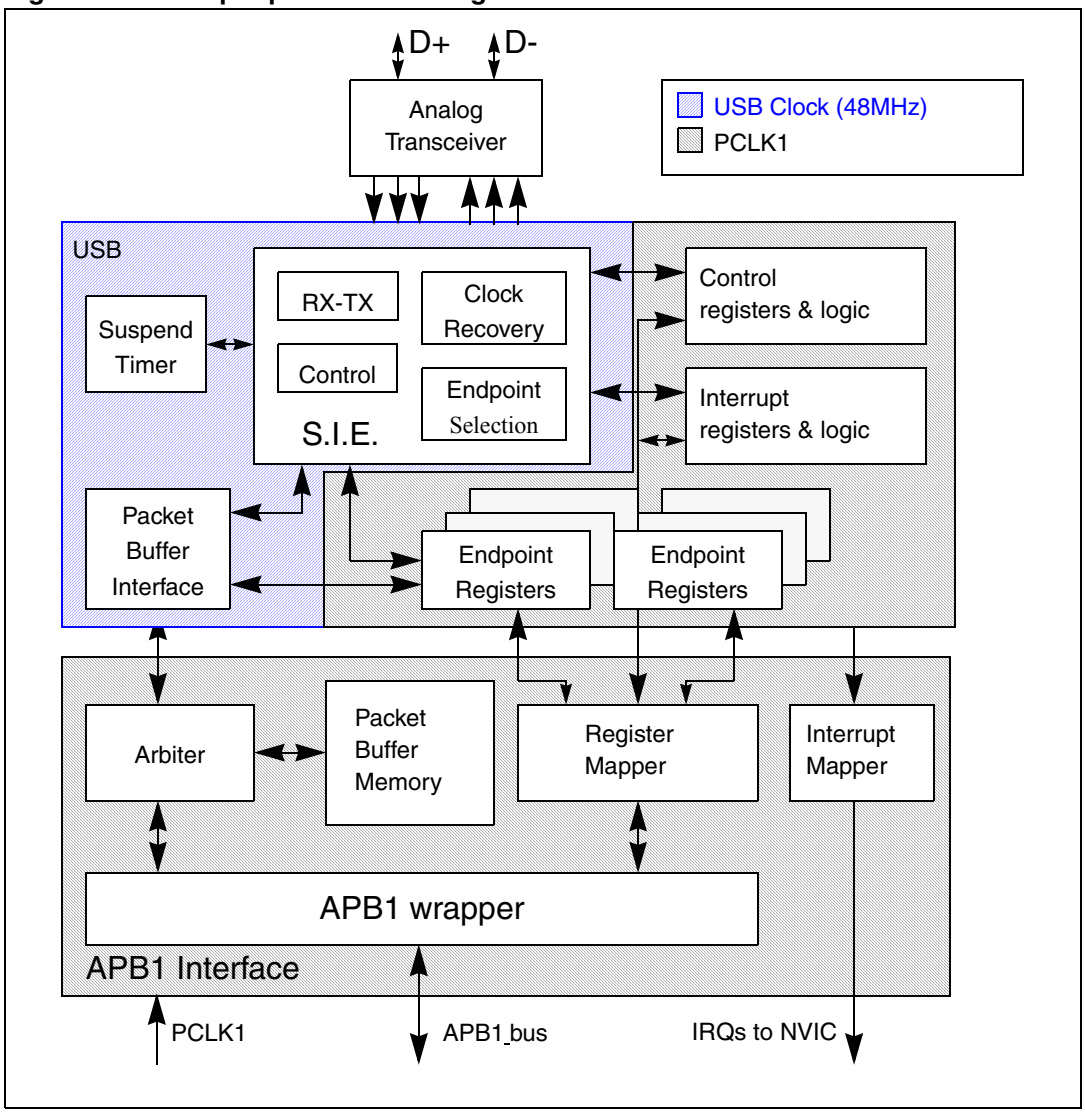
- USB specification version 2.0 full-speed compliant
- Configurable number of endpoints from 1 to 8
- Cyclic redundancy check (CRC) generation/checking, Non-return-to-zero Inverted (NRZI) encoding/decoding and bit-stuffing
- Isochronous transfers support
- Double-buffered bulk/isochronous endpoint support
- USB Suspend/Resume operations
- Frame locked clock pulse generation

Note: The USB and CAN share a dedicated 512-byte SRAM memory for data transmission and reception, and so they cannot be used concurrently (the shared SRAM is accessed through CAN and USB exclusively). The USB and CAN can be used in the same application but not at the same time.

20.3 USB functional description

Figure 188 shows the block diagram of the USB peripheral.

Figure 188. USB peripheral block diagram



The USB peripheral provides an USB compliant connection between the host PC and the function implemented by the microcontroller. Data transfer between the host PC and the system memory occurs through a dedicated packet buffer memory accessed directly by the USB peripheral. The size of this dedicated buffer memory must be according to the number of endpoints used and the maximum packet size. This dedicated memory is sized to 512 bytes and up to 16 mono-directional or 8 bidirectional endpoints can be used.The USB peripheral interfaces with the USB host, detecting token packets, handling data transmission/reception, and processing handshake packets as required by the USB standard. Transaction formatting is performed by the hardware, including CRC generation and checking.

Each endpoint is associated with a buffer description block indicating where the endpoint related memory area is located, how large it is or how many bytes must be transmitted. When a token for a valid function/endpoint pair is recognized by the USB peripheral, the related data transfer (if required and if the endpoint is configured) takes place. The data buffered by the USB peripheral is loaded in an internal 16 bit register and memory access to the dedicated buffer is performed. When all the data has been transferred, if needed, the

proper handshake packet over the USB is generated or expected according to the direction of the transfer.

At the end of the transaction, an endpoint-specific interrupt is generated, reading status registers and/or using different interrupt response routines. The microcontroller can determine:

- Which endpoint has to be served
- Which type of transaction took place, if errors occurred (bit stuffing, format, CRC, protocol, missing ACK, over/underrun, etc.)

Special support is offered to Isochronous transfers and high throughput bulk transfers, implementing a double buffer usage, which allows to always have an available buffer for the USB peripheral while the microcontroller uses the other one.

The unit can be placed in low-power mode (SUSPEND mode), by writing in the control register, whenever required. At this time, all static power dissipation is avoided, and the USB clock can be slowed down or stopped. The detection of activity at the USB inputs, while in low-power mode, wakes the device up asynchronously. A special interrupt source can be connected directly to a wakeup line to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

20.3.1 Description of USB blocks

The USB peripheral implements all the features related to USB interfacing, which include the following blocks:

- Serial Interface Engine (SIE): The functions of this block include: synchronization pattern recognition, bit-stuffing, CRC generation and checking, PID verification/generation, and handshake evaluation. It must interface with the USB transceivers and uses the virtual buffers provided by the packet buffer interface for local data storage,. This unit also generates signals according to USB peripheral events, such as Start of Frame (SOF), USB_Reset, Data errors etc. and to Endpoint related events like end of transmission or correct reception of a packet; these signals are then used to generate interrupts.
- Timer: This block generates a start-of-frame locked clock pulse and detects a global suspend (from the host) when no traffic has been received for 3 ms.
- Packet Buffer Interface: This block manages the local memory implementing a set of buffers in a flexible way, both for transmission and reception. It can choose the proper buffer according to requests coming from the SIE and locate them in the memory addresses pointed by the Endpoint registers. It increments the address after each exchanged word until the end of packet, keeping track of the number of exchanged bytes and preventing the buffer to overrun the maximum capacity.
- Endpoint-Related Registers: Each endpoint has an associated register containing the endpoint type and its current status. For mono-directional/single-buffer endpoints, a single register can be used to implement two distinct endpoints. The number of registers is 8, allowing up to 16 mono-directional/single-buffer or up to 7 double-buffer endpoints* in any combination. For example the USB peripheral can be programmed to have 4 double buffer endpoints and 8 single-buffer/mono-directional endpoints.

- Control Registers: These are the registers containing information about the status of the whole USB peripheral and used to force some USB events, such as resume and power-down.
- Interrupt Registers: These contain the Interrupt masks and a record of the events. They can be used to inquire an interrupt reason, the interrupt status or to clear the status of a pending interrupt.

*Note: * Endpoint 0 is always used for control transfer in single-buffer mode.*

The USB peripheral is connected to the APB1 bus through an APB1 interface, containing the following blocks:

- Packet Memory: This is the local memory that physically contains the Packet Buffers. It can be used by the Packet Buffer interface, which creates the data structure and can be accessed directly by the application software. The size of the Packet Memory is 512 bytes, structured as 256 words by 16 bits.
- Arbiter: This block accepts memory requests coming from the APB1 bus and from the USB interface. It resolves the conflicts by giving priority to APB1 accesses, while always reserving half of the memory bandwidth to complete all USB transfers. This time-duplex scheme implements a virtual dual-port SRAM that allows memory access, while an USB transaction is happening. Multiword APB1 transfers of any length are also allowed by this scheme.
- Register Mapper: This block collects the various byte-wide and bit-wide registers of the USB peripheral in a structured 16-bit wide word set addressed by the APB1.
- APB1 Wrapper: This provides an interface to the APB1 for the memory and register. It also maps the whole USB peripheral in the APB1 address space.
- Interrupt Mapper: This block is used to select how the possible USB events can generate interrupts and map them to three different lines of the NVIC:
  - USB low-priority interrupt (Channel 20): Triggered by all USB events (Correct transfer, USB reset, etc.). The firmware has to check the interrupt source before serving the interrupt.
  - USB high-priority interrupt (Channel 19): Triggered only by a correct transfer event for isochronous and double-buffer bulk transfer to reach the highest possible transfer rate.
  - USB wakeup interrupt (Channel 42): Triggered by the wakeup event from the USB Suspend mode.

20.4 Programming considerations

In the following sections, the expected interactions between the USB peripheral and the application program are described, in order to ease application software development.

20.4.1 Generic USB device programming

This part describes the main tasks required of the application software in order to obtain USB compliant behavior. The actions related to the most general USB events are taken into account and paragraphs are dedicated to the special cases of double-buffered endpoints and Isochronous transfers. Apart from system reset, action is always initiated by the USB peripheral, driven by one of the USB events described below.

20.4.2 System and power-on reset

Upon system and power-on reset, the first operation the application software should perform is to provide all required clock signals to the USB peripheral and subsequently de-assert its reset signal so to be able to access its registers. The whole initialization sequence is hereafter described.

As a first step application software needs to activate register macrocell clock and de-assert macrocell specific reset signal using related control bits provided by device clock management logic.

After that, the analog part of the device related to the USB transceiver must be switched on using the PDWN bit in CNTR register, which requires a special handling. This bit is intended to switch on the internal voltage references that supply the port transceiver. This circuit has a defined startup time ($t_{STARTUP}$ specified in the datasheet) during which the behavior of the USB transceiver is not defined. It is thus necessary to wait this time, after setting the PDWN bit in the CNTR register, before removing the reset condition on the USB part (by clearing the FRES bit in the CNTR register). Clearing the ISTR register then removes any spurious pending interrupt before any other macrocell operation is enabled.

At system reset, the microcontroller must initialize all required registers and the packet buffer description table, to make the USB peripheral able to properly generate interrupts and data transfers. All registers not specific to any endpoint must be initialized according to the needs of application software (choice of enabled interrupts, chosen address of packet buffers, etc.). Then the process continues as for the USB reset case (see further paragraph).

USB reset (RESET interrupt)

When this event occurs, the USB peripheral is put in the same conditions it is left by the system reset after the initialization described in the previous paragraph: communication is disabled in all endpoint registers (the USB peripheral will not respond to any packet). As a response to the USB reset event, the USB function must be enabled, having as USB address 0, implementing only the default control endpoint (endpoint address is 0 too). This is accomplished by setting the Enable Function (EF) bit of the USB_DADDR register and initializing the EP0R register and its related packet buffers accordingly. During USB enumeration process, the host assigns a unique address to this device, which must be written in the ADD[6:0] bits of the USB_DADDR register, and configures any other necessary endpoint.
When a RESET interrupt is received, the application software is responsible to enable again the default endpoint of USB function 0 within 10mS from the end of reset sequence which triggered the interrupt.

Structure and usage of packet buffers

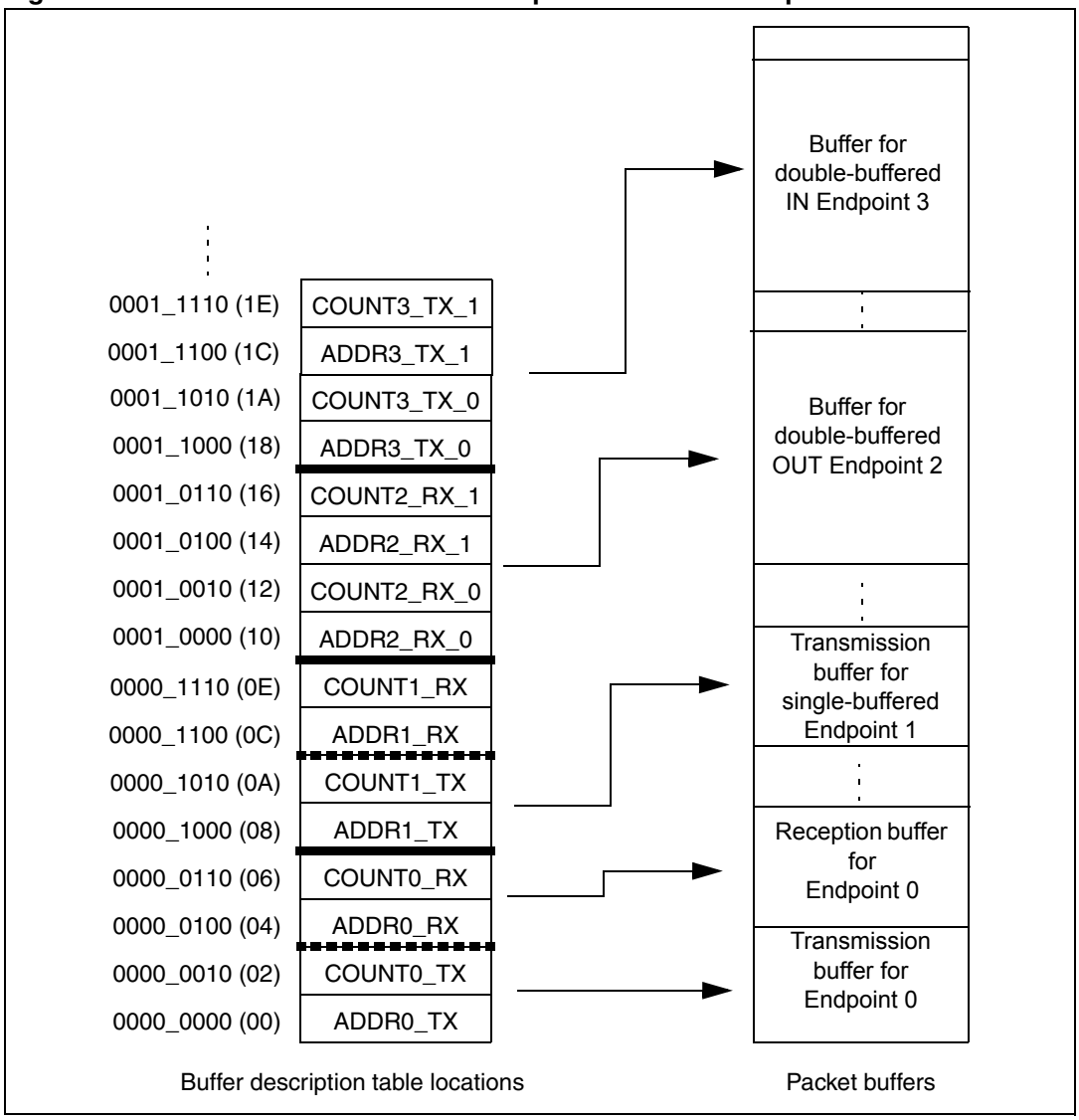
Each bidirectional endpoint may receive or transmit data from/to the host. The received data is stored in a dedicated memory buffer reserved for that endpoint, while another memory buffer contains the data to be transmitted by the endpoint. Access to this memory is performed by the packet buffer interface block, which delivers a memory access request and waits for its acknowledgement. Since the packet buffer memory has to be accessed by the microcontroller also, an arbitration logic takes care of the access conflicts, using half APB1 cycle for microcontroller access and the remaining half for the USB peripheral access. In this way, both the agents can operate as if the packet memory is a dual-port SRAM, without being aware of any conflict even when the microcontroller is performing back-to-back accesses. The USB peripheral logic uses a dedicated clock. The frequency of this dedicated

clock is fixed by the requirements of the USB standard at 48 MHz, and this can be different from the clock used for the interface to the APB1 bus. Different clock configurations are possible where the APB1 clock frequency can be higher or lower than the USB peripheral one.

Note: Due to USB data rate and packet memory interface requirements, the APB1 clock frequency must be greater than 8 MHz to avoid data overrun/underrun problems.

Each endpoint is associated with two packet buffers (usually one for transmission and the other one for reception). Buffers can be placed anywhere inside the packet memory because their location and size is specified in a buffer description table, which is also located in the packet memory at the address indicated by the USB_BTABLE register. Each table entry is associated to an endpoint register and it is composed of four 16-bit words so that table start address must always be aligned to an 8-byte boundary (the lowest three bits of USB_BTABLE register are always “000”). Buffer descriptor table entries are described in the *Section 20.5.3: Buffer descriptor table*. If an endpoint is unidirectional and it is neither an Isochronous nor a double-buffered bulk, only one packet buffer is required (the one related to the supported transfer direction). Other table locations related to unsupported transfer directions or unused endpoints, are available to the user. isochronous and double-buffered bulk endpoints have special handling of packet buffers (Refer to *Section 20.4.4: Isochronous transfers* and *Section 20.4.3: Double-buffered endpoints* respectively). The relationship between buffer description table entries and packet buffer areas is depicted in *Figure 189*.

Figure 189. Packet buffer areas with examples of buffer description table locations



Each packet buffer is used either during reception or transmission starting from the bottom. The USB peripheral will never change the contents of memory locations adjacent to the allocated memory buffers; if a packet bigger than the allocated buffer length is received (buffer overrun condition) the data will be copied to the memory only up to the last available location.

Endpoint initialization

The first step to initialize an endpoint is to write appropriate values to the ADDRn_TX/ADDRn_RX registers so that the USB peripheral finds the data to be transmitted already available and the data to be received can be buffered. The EP_TYPE bits in the USB_EPnR register must be set according to the endpoint type, eventually using the EP_KIND bit to enable any special required feature. On the transmit side, the endpoint must be enabled using the STAT_TX bits in the USB_EPnR register and COUNTn_TX must be initialized. For reception, STAT_RX bits must be set to enable reception and COUNTn_RX must be written with the allocated buffer size using the BL_SIZE and

NUM_BLOCK fields. Unidirectional endpoints, except Isochronous and double-buffered bulk endpoints, need to initialize only bits and registers related to the supported direction. Once the transmission and/or reception are enabled, register USB_EPnR and locations ADDRn_TX/ADDRn_RX, COUNTn_TX/COUNTn_RX (respectively), should not be modified by the application software, as the hardware can change their value on the fly. When the data transfer operation is completed, notified by a CTR interrupt event, they can be accessed again to re-enable a new operation.

IN packets (data transmission)

When receiving an IN token packet, if the received address matches a configured and valid endpoint one, the USB peripheral accesses the contents of ADDRn_TX and COUNTn_TX locations inside buffer descriptor table entry related to the addressed endpoint. The content of these locations is stored in its internal 16 bit registers ADDR and COUNT (not accessible by software). The packet memory is accessed again to read the first word to be transmitted (Refer to *Structure and usage of packet buffers on page 462*) and starts sending a DATA0 or DATA1 PID according to USB_EPnR bit DTOG_TX. When the PID is completed, the first byte from the word, read from buffer memory, is loaded into the output shift register to be transmitted on the USB bus. After the last data byte is transmitted, the computed CRC is sent. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the data packet, according to STAT_TX bits in the USB_EPnR register.

The ADDR internal register is used as a pointer to the current buffer memory location while COUNT is used to count the number of remaining bytes to be transmitted. Each word read from the packet buffer memory is transmitted over the USB bus starting from the least significant byte. Transmission buffer memory is read starting from the address pointed by ADDRn_TX for COUNTn_TX/2 words. If a transmitted packet is composed of an odd number of bytes, only the lower half of the last word accessed will be used.

On receiving the ACK receipt by the host, the USB_EPnR register is updated in the following way: DTOG_TX bit is toggled, the endpoint is made invalid by setting STAT_TX=10 (NAK) and bit CTR_TX is set. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. Servicing of the CTR_TX event starts clearing the interrupt bit; the application software then prepares another buffer full of data to be sent, updates the COUNTn_TX table location with the number of byte to be transmitted during the next transfer, and finally sets STAT_TX to ‘11’ (VALID) to re-enable transmissions. While the STAT_TX bits are equal to ‘10’ (NAK), any IN request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second IN transaction addressed to the same endpoint immediately following the one which triggered the CTR interrupt.

OUT and SETUP packets (data reception)

These two tokens are handled by the USB peripheral more or less in the same way; the differences in the handling of SETUP packets are detailed in the following paragraph about control transfers. When receiving an OUT/SETUP PID, if the address matches a valid endpoint, the USB peripheral accesses the contents of the ADDRn_RX and COUNTn_RX locations inside the buffer descriptor table entry related to the addressed endpoint. The content of the ADDRn_RX is stored directly in its internal register ADDR. While COUNT is now reset and the values of BL_SIZE and NUM_BLOCK bit fields, which are read within COUNTn_RX content are used to initialize BUF_COUNT, an internal 16 bit counter, which is used to check the buffer overrun condition (all these internal registers are not accessible by

software). Data bytes subsequently received by the USB peripheral are packed in words (the first byte received is stored as least significant byte) and then transferred to the packet buffer starting from the address contained in the internal ADDR register while BUF_COUNT is decremented and COUNT is incremented at each byte transfer. When the end of DATA packet is detected, the correctness of the received CRC is tested and only if no errors occurred during the reception, an ACK handshake packet is sent back to the transmitting host. In case of wrong CRC or other kinds of errors (bit-stuff violations, frame errors, etc.), data bytes are still copied in the packet memory buffer, at least until the error detection point, but ACK packet is not sent and the ERR bit in USB_ISTR register is set. However, there is usually no software action required in this case: the USB peripheral recovers from reception errors and remains ready for the next transaction to come. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the ACK, according to bits STAT_RX in the USB_EPnR register and no data is written in the reception memory buffers.

Reception memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC included (i.e. data payload length + 2), or up to the last allocated memory location, as defined by BL_SIZE and NUM_BLOCK, whichever comes first. In this way, the USB peripheral never writes beyond the end of the allocated reception memory buffer area. If the length of the data packet payload (actual number of bytes used by the application) is greater than the allocated buffer, the USB peripheral detects a buffer overrun condition. in this case, a STALL handshake is sent instead of the usual ACK to notify the problem to the host, no interrupt is generated and the transaction is considered failed.

When the transaction is completed correctly, by sending the ACK handshake packet, the internal COUNT register is copied back in the COUNTn_RX location inside the buffer description table entry, leaving unaffected BL_SIZE and NUM_BLOCK fields, which normally do not require to be re-written, and the USB_EPnR register is updated in the following way: DTOG_RX bit is toggled, the endpoint is made invalid by setting STAT_RX = ‘10’ (NAK) and bit CTR_RX is set. If the transaction has failed due to errors or buffer overrun condition, none of the previously listed actions take place. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. The CTR_RX event is serviced by first determining the transaction type (SETUP bit in the USB_EPnR register); the application software must clear the interrupt flag bit and get the number of received bytes reading the COUNTn_RX location inside the buffer description table entry related to the endpoint being processed. After the received data is processed, the application software should set the STAT_RX bits to ‘11’ (Valid) in the USB_EPnR, enabling further transactions. While the STAT_RX bits are equal to ‘10’ (NAK), any OUT request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second OUT transaction addressed to the same endpoint following immediately the one which triggered the CTR interrupt.

Control transfers

Control transfers are made of a SETUP transaction, followed by zero or more data stages, all of the same direction, followed by a status stage (a zero-byte transfer in the opposite direction). SETUP transactions are handled by control endpoints only and are very similar to OUT ones (data reception) except that the values of DTOG_TX and DTOG_RX bits of the addressed endpoint registers are set to 1 and 0 respectively, to initialize the control transfer, and both STAT_TX and STAT_RX are set to ‘10’ (NAK) to let software decide if subsequent transactions must be IN or OUT depending on the SETUP contents. A control endpoint must check SETUP bit in the USB_EPnR register at each CTR_RX event to distinguish normal

OUT transactions from SETUP ones. A USB device can determine the number and direction of data stages by interpreting the data transferred in the SETUP stage, and is required to STALL the transaction in the case of errors. To do so, at all data stages before the last, the unused direction should be set to STALL, so that, if the host reverses the transfer direction too soon, it gets a STALL as a status stage. While enabling the last data stage, the opposite direction should be set to NAK, so that, if the host reverses the transfer direction (to perform the status stage) immediately, it is kept waiting for the completion of the control operation. If the control operation completes successfully, the software will change NAK to VALID, otherwise to STALL. At the same time, if the status stage will be an OUT, the STATUS_OUT (EP_KIND in the USB_EPnR register) bit should be set, so that an error is generated if a status transaction is performed with not-zero data. When the status transaction is serviced, the application clears the STATUS_OUT bit and sets STAT_RX to VALID (to accept a new command) and STAT_TX to NAK (to delay a possible status stage immediately following the next setup).

Since the USB specification states that a SETUP packet cannot be answered with a handshake different from ACK, eventually aborting a previously issued command to start the new one, the USB logic doesn't allow a control endpoint to answer with a NAK or STALL packet to a SETUP token received from the host.

When the STAT_RX bits are set to '01' (STALL) or '10' (NAK) and a SETUP token is received, the USB accepts the data, performing the required data transfers and sends back an ACK handshake. If that endpoint has a previously issued CTR_RX request not yet acknowledged by the application (i.e. CTR_RX bit is still set from a previously completed reception), the USB discards the SETUP transaction and does not answer with any handshake packet regardless of its state, simulating a reception error and forcing the host to send the SETUP token again. This is done to avoid losing the notification of a SETUP transaction addressed to the same endpoint immediately following the transaction, which triggered the CTR_RX interrupt.

20.4.3 Double-buffered endpoints

All different endpoint types defined by the USB standard represent different traffic models, and describe the typical requirements of different kind of data transfer operations. When large portions of data are to be transferred between the host PC and the USB function, the bulk endpoint type is the most suited model. This is because the host schedules bulk transactions so as to fill all the available bandwidth in the frame, maximizing the actual transfer rate as long as the USB function is ready to handle a bulk transaction addressed to it. If the USB function is still busy with the previous transaction when the next one arrives, it will answer with a NAK handshake and the host PC will issue the same transaction again until the USB function is ready to handle it, reducing the actual transfer rate due to the bandwidth occupied by re-transmissions. For this reason, a dedicated feature called 'double-buffering' can be used with bulk endpoints.

When 'double-buffering' is activated, data toggle sequencing is used to select, which buffer is to be used by the USB peripheral to perform the required data transfers, using both 'transmission' and 'reception' packet memory areas to manage buffer swapping on each successful transaction in order to always have a complete buffer to be used by the application, while the USB peripheral fills the other one. For example, during an OUT transaction directed to a 'reception' double-buffered bulk endpoint, while one buffer is being filled with new data coming from the USB host, the other one is available for the microcontroller software usage (the same would happen with a 'transmission' double-buffered bulk endpoint and an IN transaction).

Since the swapped buffer management requires the usage of all 4 buffer description table locations hosting the address pointer and the length of the allocated memory buffers, the USB_EPnR registers used to implement double-buffered bulk endpoints are forced to be used as unidirectional ones. Therefore, only one STAT bit pair must be set at a value different from '00' (Disabled): STAT_RX if the double-buffered bulk endpoint is enabled for reception, STAT_TX if the double-buffered bulk endpoint is enabled for transmission. In case it is required to have double-buffered bulk endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

To exploit the double-buffering feature and reach the highest possible transfer rate, the endpoint flow control structure, described in previous chapters, has to be modified, in order to switch the endpoint status to NAK only when a buffer conflict occurs between the USB peripheral and application software, instead of doing it at the end of each successful transaction. The memory buffer which is currently being used by the USB peripheral is defined by the DTOG bit related to the endpoint direction: DTOG_RX (bit 14 of USB_EPnR register) for 'reception' double-buffered bulk endpoints or DTOG_TX (bit 6 of USB_EPnR register) for 'transmission' double-buffered bulk endpoints. To implement the new flow control scheme, the USB peripheral should know which packet buffer is currently in use by the application software, so to be aware of any conflict. Since in the USB_EPnR register, there are two DTOG bits but only one is used by USB peripheral for data and buffer sequencing (due to the unidirectional constraint required by double-buffering feature) the other one can be used by the application software to show which buffer it is currently using. This new buffer flag is called SW_BUF. In the following table the correspondence between USB_EPnR register bits and DTOG/SW_BUF definition is explained, for the cases of 'transmission' and 'reception' double-buffered bulk endpoints.

Table 134. Double-buffering buffer flag definition

Buffer flag	'Transmission' endpoint	'Reception' endpoint
DTOG	DTOG_TX (USB_EPnRbit 6)	DTOG_RX (USB_EPnRbit 14)
SW_BUF	USB_EPnR bit 14	USB_EPnR bit 6

The memory buffer which is currently being used by the USB peripheral is defined by DTOG buffer flag, while the buffer currently in use by application software is identified by SW_BUF buffer flag. The relationship between the buffer flag value and the used packet buffer is the same in both cases, and it is listed in the following table.

Table 135. Bulk double-buffering memory buffers usage

Endpoint Type	DTOG	SW_BUF	Packet buffer used by USB Peripheral	Packet buffer used by Application Software
IN	0	1	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.	ADDRn_TX_1 / COUNTn_TX_1 Buffer description table locations.
	1	0	ADDRn_TX_1 / COUNTn_TX_1 Buffer description table locations	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
	0	0	None (1)	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
	1	1	None (1)	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
OUT	0	1	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.
	1	0	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.
	0	0	None (1)	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.
	1	1	None (1)	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.

1. Endpoint in NAK Status.

Double-buffering feature for a bulk endpoint is activated by:

- Writing EP_TYPE bit field at '00' in its USB_EPnR register, to define the endpoint as a bulk, and
- Setting EP_KIND bit at '1' (DBL_BUF), in the same register.

The application software is responsible for DTOG and SW_BUF bits initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. The end of the first transaction occurring after having set DBL_BUF, triggers the special flow control of double-buffered bulk endpoints, which is used for all other transactions addressed to this endpoint until DBL_BUF remain set. At the end of each transaction the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making the USB peripheral buffer swapping completely software independent. Unlike common transactions, and the first one after DBL_BUF setting, STAT bit pair is not affected by the transaction termination and its value remains '11' (Valid). However, as the token packet of a new transaction is received, the

actual endpoint status will be masked as ‘10’ (NAK) when a buffer conflict between the USB peripheral and the application software is detected (this condition is identified by DTOG and SW_BUF having the same value, see *Table 135 on page 469*). The application software responds to the CTR event notification by clearing the interrupt flag and starting any required handling of the completed transaction. When the application packet buffer usage is over, the software toggles the SW_BUF bit, writing ‘1’ to it, to notify the USB peripheral about the availability of that buffer. In this way, the number of NAKed transactions is limited only by the application elaboration time of a transaction data: if the elaboration time is shorter than the time required to complete a transaction on the USB bus, no re-transmissions due to flow control will take place and the actual transfer rate will be limited only by the host PC.

The application software can always override the special flow control implemented for double-buffered bulk endpoints, writing an explicit status different from ‘11’ (Valid) into the STAT bit pair of the related USB_EPnR register. In this case, the USB peripheral will always use the programmed endpoint status, regardless of the buffer usage condition.

20.4.4 Isochronous transfers

The USB standard supports full speed peripherals requiring a fixed and accurate data production/consume frequency, defining this kind of traffic as ‘Isochronous’. Typical examples of this data are: audio samples, compressed video streams, and in general any sort of sampled data having strict requirements for the accuracy of delivered frequency. When an endpoint is defined to be ‘isochronous’ during the enumeration phase, the host allocates in the frame the required bandwidth and delivers exactly one IN or OUT packet each frame, depending on endpoint direction. To limit the bandwidth requirements, no re-transmission of failed transactions is possible for Isochronous traffic; this leads to the fact that an isochronous transaction does not have a handshake phase and no ACK packet is expected or sent after the data packet. For the same reason, Isochronous transfers do not support data toggle sequencing and always use DATA0 PID to start any data packet.

The Isochronous behavior for an endpoint is selected by setting the EP_TYPE bits at ‘10’ in its USB_EPnR register; since there is no handshake phase the only legal values for the STAT_RX/STAT_TX bit pairs are ‘00’ (Disabled) and ‘11’ (Valid), any other value will produce results not compliant to USB standard. Isochronous endpoints implement double-buffering to ease application software development, using both ‘transmission’ and ‘reception’ packet memory areas to manage buffer swapping on each successful transaction in order to have always a complete buffer to be used by the application, while the USB peripheral fills the other.

The memory buffer which is currently used by the USB peripheral is defined by the DTOG bit related to the endpoint direction (DTOG_RX for ‘reception’ isochronous endpoints, DTOG_TX for ‘transmission’ isochronous endpoints, both in the related USB_EPnR register) according to *Table 136*.

Table 136. Isochronous memory buffers usage

Endpoint Type	DTOG bit value	Packet buffer used by the USB peripheral	Packet buffer used by the application software
IN	0	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.
	1	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.
OUT	0	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.
	1	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.

As it happens with double-buffered bulk endpoints, the USB_EPnR registers used to implement Isochronous endpoints are forced to be used as unidirectional ones. In case it is required to have Isochronous endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

The application software is responsible for the DTOG bit initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. At the end of each transaction, the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making buffer swapping completely software independent. STAT bit pair is not affected by transaction completion; since no flow control is possible for Isochronous transfers due to the lack of handshake phase, the endpoint remains always ‘11’ (Valid). CRC errors or buffer-overrun conditions occurring during Isochronous OUT transfers are anyway considered as correct transactions and they always trigger an CTR_RX event. However, CRC errors will anyway set the ERR bit in the USB_ISTR register to notify the software of the possible data corruption.

20.4.5 Suspend/Resume events

The USB standard defines a special peripheral state, called SUSPEND, in which the average current drawn from the USB bus must not be greater than 500 µA. This requirement is of fundamental importance for bus-powered devices, while self-powered devices are not required to comply to this strict power consumption constraint. In suspend mode, the host PC sends the notification to not send any traffic on the USB bus for more than 3mS: since a SOF packet must be sent every mS during normal operations, the USB peripheral detects the lack of 3 consecutive SOF packets as a suspend request from the host PC and set the SUSP bit to ‘1’ in USB_ISTR register, causing an interrupt if enabled. Once the device is suspended, its normal operation can be restored by a so called RESUME sequence, which can be started from the host PC or directly from the peripheral itself, but it is always terminated by the host PC. The suspended USB peripheral must be anyway able to detect a RESET sequence, reacting to this event as a normal USB reset event.

The actual procedure used to suspend the USB peripheral is device dependent since according to the device composition, different actions may be required to reduce the total consumption.

A brief description of a typical suspend procedure is provided below, focused on the USB-related aspects of the application software routine responding to the SUSP notification of the USB peripheral:

1. Set the FSUSP bit in the USB_CNTR register to 1. This action activates the suspend mode within the USB peripheral. As soon as the suspend mode is activated, the check on SOF reception is disabled to avoid any further SUSP interrupts being issued while the USB is suspended.
2. Remove or reduce any static power consumption in blocks different from the USB peripheral.
3. Set LP_MODE bit in USB_CNTR register to 1 to remove static power consumption in the analog USB transceivers but keeping them able to detect resume activity.
4. Optionally turn off external oscillator and device PLL to stop any activity inside the device.

When an USB event occurs while the device is in SUSPEND mode, the RESUME procedure must be invoked to restore nominal clocks and regain normal USB behavior. Particular care must be taken to insure that this process does not take more than 10mS when the wakening event is an USB reset sequence (See “Universal Serial Bus Specification” for more details). The start of a resume or reset sequence, while the USB peripheral is suspended, clears the LP_MODE bit in USB_CNTR register asynchronously. Even if this event can trigger an WKUP interrupt if enabled, the use of an interrupt response routine must be carefully evaluated because of the long latency due to system clock restart; to have the shorter latency before re-activating the nominal clock it is suggested to put the resume procedure just after the end of the suspend one, so its code is immediately executed as soon as the system clock restarts. To prevent ESD discharges or any other kind of noise from waking-up the system (the exit from suspend mode is an asynchronous event), a suitable analog filter on data line status is activated during suspend; the filter width is about 70ns.

The following is a list of actions a resume procedure should address:

1. Optionally turn on external oscillator and/or device PLL.
2. Clear FSUSP bit of USB_CNTR register.
3. If the resume triggering event has to be identified, bits RXDP and RXDM in the USB_FNR register can be used according to *Table 137*, which also lists the intended software action in all the cases. If required, the end of resume or reset sequence can be detected monitoring the status of the above mentioned bits by checking when they reach the “10” configuration, which represent the Idle bus state; moreover at the end of a reset sequence the RESET bit in USB_ISTR register is set to 1, issuing an interrupt if enabled, which should be handled as usual.

Table 137. Resume event detection

[RXDP,RXDM] Status	Wakeup event	Required resume software action
“00”	Root reset	None
“10”	None (noise on bus)	Go back in Suspend mode
“01”	Root resume	None
“11”	Not Allowed (noise on bus)	Go back in Suspend mode

A device may require to exit from suspend mode as an answer to particular events not directly related to the USB protocol (e.g. a mouse movement wakes up the whole system).

In this case, the resume sequence can be started by setting the RESUME bit in the USB_CNTR register to '1' and resetting it to 0 after an interval between 1mS and 15mS (this interval can be timed using ESOF interrupts, occurring with a 1mS period when the system clock is running at nominal frequency). Once the RESUME bit is clear, the resume sequence will be completed by the host PC and its end can be monitored again using the RXDP and RXDM bits in the USB_FNR register.

Note: The RESUME bit must be anyway used only after the USB peripheral has been put in suspend mode, setting the FSUSP bit in USB_CNTR register to 1.

20.5 USB registers

The USB peripheral registers can be divided into the following groups:

- Common Registers: Interrupt and Control registers
- Endpoint Registers: Endpoint configuration and status
- Buffer Descriptor Table: Location of packet memory used to locate data buffers

All register addresses are expressed as offsets with respect to the USB peripheral registers base address 0x4000 5C00, except the buffer descriptor table locations, which starts at the address specified by the USB_BTABLE register. Due to the common limitation of APB1 bridges on word addressability, all register addresses are aligned to 32-bit word boundaries although they are 16-bit wide. The same address alignment is used to access packet buffer memory locations, which are located starting from 0x4000 6000.

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

20.5.1 Common registers

These registers affect the general behavior of the USB peripheral defining operating mode, interrupt handling, device address and giving access to the current frame number updated by the host PC.

USB control register (USB_CNTR)

Address offset: 0x40

Reset value: 0x0003

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRM	PMAOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LP_MODE	PDWN	FRES
rw	rw	rw	rw	rw	rw	rw	rw	Res.			rw	rw	rw	rw	rw

Bit 15 **CTRM:** *Correct Transfer Interrupt Mask*

0: Correct Transfer (CTR) Interrupt disabled.
1: CTR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 14 **PMAOVRM:** *Packet Memory Area Over / Underrun Interrupt Mask*

0: PMAOVR Interrupt disabled.
1: PMAOVR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 13 **ERRM:** *Error Interrupt Mask*

0: ERR Interrupt disabled.
1: ERR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 12 **WKUPM:** *Wakeup Interrupt Mask*

0: WKUP Interrupt disabled.
1: WKUP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 11 **SUSPM:** *Suspend mode Interrupt Mask*

0: Suspend Mode Request (SUSP) Interrupt disabled.
1: SUSP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 10 **RESETM:** *USB Reset Interrupt Mask*

0: RESET Interrupt disabled.
1: RESET Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 9 **SOFM:** *Start Of Frame Interrupt Mask*

0: SOF Interrupt disabled.
1: SOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 8 **ESOFM:** *Expected Start Of Frame Interrupt Mask*

0: Expected Start of Frame (ESOF) Interrupt disabled.
1: ESOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bits 7:5 Reserved.

Bit 4 **RESUME:** *Resume request*

The microcontroller can set this bit to send a Resume signal to the host. It must be activated, according to USB specifications, for no less than 1mS and no more than 15mS after which the Host PC is ready to drive the resume sequence up to its end.

Bit 3 **FSUSP:** *Force suspend*

Software must set this bit when the SUSP interrupt is received, which is issued when no traffic is received by the USB peripheral for 3 mS.
0: No effect.
1: Enter suspend mode. Clocks and static power dissipation in the analog transceiver are left unaffected. If suspend power consumption is a requirement (bus-powered device), the application software should set the LP_MODE bit after FSUSP as explained below.

Bit 2 **LP_MODE:** *Low-power mode*

This mode is used when the suspend-mode power constraints require that all static power dissipation is avoided, except the one required to supply the external pull-up resistor. This condition should be entered when the application is ready to stop all system clocks, or reduce their frequency in order to meet the power consumption requirements of the USB suspend condition. The USB activity during the suspend mode (WKUP event) asynchronously resets this bit (it can also be reset by software).
0: No Low-power mode.
1: Enter Low-power mode.

Bit 1 **PDWN:** *Power down*

This bit is used to completely switch off all USB-related analog parts if it is required to completely disable the USB peripheral for any reason. When this bit is set, the USB peripheral is disconnected from the transceivers and it cannot be used.
0: Exit Power Down.
1: Enter Power down mode.

Bit 0 **FRES:** *Force USB Reset*

0: Clear USB reset.
1: Force a reset of the USB peripheral, exactly like a RESET signalling on the USB. The USB peripheral is held in RESET state until software clears this bit. A “USB-RESET” interrupt is generated, if enabled.

USB interrupt status register (USB_ISTR)

Address offset: 0x44

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR	PMA OVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved			DIR	EP_ID[3:0]			
r	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	Res.			r	r	r	r	r

This register contains the status of all the interrupt sources allowing application software to determine, which events caused an interrupt request.

The upper part of this register contains single bits, each of them representing a specific event. These bits are set by the hardware when the related event occurs; if the corresponding bit in the USB_CNTR register is set, a generic interrupt request is generated. The interrupt routine, examining each bit, will perform all necessary actions, and finally it will clear the serviced bits. If any of them is not cleared, the interrupt is considered to be still pending, and the interrupt line will be kept high again. If several bits are set simultaneously, only a single interrupt will be generated.

Endpoint transaction completion can be handled in a different way to reduce interrupt response latency. The CTR bit is set by the hardware as soon as an endpoint successfully completes a transaction, generating a generic interrupt request if the corresponding bit in USB_CNTR is set. An endpoint dedicated interrupt condition is activated independently from the CTRM bit in the USB_CNTR register. Both interrupt conditions remain active until software clears the pending bit in the corresponding USB_EPnR register (the CTR bit is actually a read only bit). For endpoint-related interrupts, the software can use the Direction of Transaction (DIR) and EP_ID read-only bits to identify, which endpoint made the last interrupt request and called the corresponding interrupt service routine.

The user can choose the relative priority of simultaneously pending USB_ISTR events by specifying the order in which software checks USB_ISTR bits in an interrupt service routine. Only the bits related to events, which are serviced, are cleared. At the end of the service routine, another interrupt will be requested, to service the remaining conditions.

To avoid spurious clearing of some bits, it is recommended to clear them with a load instruction where all bits which must not be altered are written with 1, and all bits to be cleared are written with ‘0’ (these bits can only be cleared by software). Read-modify-write cycles should be avoided because between the read and the write operations another bit could be set by the hardware and the next write will clear it before the microprocessor has the time to serve the event.

The following describes each bit in detail:

Bit 15 **CTR:** *Correct Transfer*

This bit is set by the hardware to indicate that an endpoint has successfully completed a transaction; using DIR and EP_ID bits software can determine which endpoint requested the interrupt. This bit is read-only.

Bit 14 **PMAOVR:** *Packet Memory Area Over / Underrun*

This bit is set if the microcontroller has not been able to respond in time to an USB memory request. The USB peripheral handles this event in the following way: During reception an ACK handshake packet is not sent, during transmission a bit-stuff error is forced on the transmitted stream; in both cases the host will retry the transaction. The PMAOVR interrupt should never occur during normal operations. Since the failed transaction is retried by the host, the application software has the chance to speed-up device operations during this interrupt handling, to be ready for the next transaction retry; however this does not happen during Isochronous transfers (no isochronous transaction is anyway retried) leading to a loss of data in this case. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 13 **ERR:** *Error*

This flag is set whenever one of the errors listed below has occurred:
NANS: No ANSwer. The timeout for a host response has expired.
CRC: Cyclic Redundancy Check error. One of the received CRCs, either in the token or in the data, was wrong.
BST: Bit Stuffing error. A bit stuffing error was detected anywhere in the PID, data, and/or CRC.
FVIO: Framing format Violation. A non-standard frame was received (EOP not in the right place, wrong token sequence, etc.).
The USB software can usually ignore errors, since the USB peripheral and the PC host manage retransmission in case of errors in a fully transparent way. This interrupt can be useful during the software development phase, or to monitor the quality of transmission over the USB bus, to flag possible problems to the user (e.g. loose connector, too noisy environment, broken conductor in the USB cable and so on). This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 12 **WKUP:** *Wake up*

This bit is set to 1 by the hardware when, during suspend mode, activity is detected that wakes up the USB peripheral. This event asynchronously clears the LP_MODE bit in the CTLR register and activates the USB_WAKEUP line, which can be used to notify the rest of the device (e.g. wakeup unit) about the start of the resume process. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 11 **SUSP:** *Suspend mode request*

This bit is set by the hardware when no traffic has been received for 3mS, indicating a suspend mode request from the USB bus. The suspend condition check is enabled immediately after any USB reset and it is disabled by the hardware when the suspend mode is active (FSUSP=1) until the end of resume sequence. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 10 **RESET:** *USB RESET request*

Set when the USB peripheral detects an active USB RESET signal at its inputs. The USB peripheral, in response to a RESET, just resets its internal protocol state machine, generating an interrupt if RESETM enable bit in the USB_CNTR register is set. Reception and transmission are disabled until the RESET bit is cleared. All configuration registers do not reset: the microcontroller must explicitly clear these registers (this is to ensure that the RESET interrupt can be safely delivered, and any transaction immediately followed by a RESET can be completed). The function address and endpoint registers are reset by an USB reset event.
This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 9 **SOF:** *Start Of Frame*

This bit signals the beginning of a new USB frame and it is set when a SOF packet arrives through the USB bus. The interrupt service routine may monitor the SOF events to have a 1mS synchronization event to the USB host and to safely read the USB_FNR register which is updated at the SOF packet reception (this could be useful for isochronous applications). This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bit 8 **ESOF:** *Expected Start Of Frame*

This bit is set by the hardware when an SOF packet is expected but not received. The host sends an SOF packet each mS, but if the hub does not receive it properly, the Suspend Timer issues this interrupt. If three consecutive ESOF interrupts are generated (i.e. three SOF packets are lost) without any traffic occurring in between, a SUSP interrupt is generated. This bit is set even when the missing SOF packets occur while the Suspend Timer is not yet locked. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bits 7:5 Reserved.

Bit 4 **DIR:** *Direction of transaction*.

This bit is written by the hardware according to the direction of the successful transaction, which generated the interrupt request.
If DIR bit=0, CTR_TX bit is set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of IN type (data transmitted by the USB peripheral to the host PC).
If DIR bit=1, CTR_RX bit or both CTR_TX/CTR_RX are set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of OUT type (data received by the USB peripheral from the host PC) or two pending transactions are waiting to be processed.
This information can be used by the application software to access the USB_EPnR bits related to the triggering transaction since it represents the direction having the interrupt pending. This bit is read-only.

Bits 3:0 **EP_ID[3:0]:** *Endpoint Identifier.*

These bits are written by the hardware according to the endpoint number, which generated the interrupt request. If several endpoint transactions are pending, the hardware writes the endpoint identifier related to the endpoint having the highest priority defined in the following way: Two endpoint sets are defined, in order of priority: Isochronous and double-buffered bulk endpoints are considered first and then the other endpoints are examined. If more than one endpoint from the same set is requesting an interrupt, the EP_ID bits in USB_ISTR register are assigned according to the lowest requesting endpoint register, EP0R having the highest priority followed by EP1R and so on. The application software can assign a register to each endpoint according to this priority scheme, so as to order the concurring endpoint requests in a suitable way. These bits are read only.

USB frame number register (USB_FNR)

Address offset: 0x48

Reset value: 0x0XXX where X is undefined

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]										
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 15 **RXDP:** *Receive Data + Line Status*

This bit can be used to observe the status of received data plus upstream port data line. It can be used during end-of-suspend routines to help determining the wakeup event.

Bit 14 **RXDM:** *Receive Data - Line Status*

This bit can be used to observe the status of received data minus upstream port data line. It can be used during end-of-suspend routines to help determining the wakeup event.

Bit 13 **LCK:** *Locked*

This bit is set by the hardware when at least two consecutive SOF packets have been received after the end of an USB reset condition or after the end of an USB resume sequence. Once locked, the frame timer remains in this state until an USB reset or USB suspend event occurs.

Bits 12:11 **LSOF[1:0]:** *Lost SOF*

These bits are written by the hardware when an ESOF interrupt is generated, counting the number of consecutive SOF packets lost. At the reception of an SOF packet, these bits are cleared.

Bits 10:0 **FN[10:0]:** *Frame Number*

This bit field contains the 11-bits frame number contained in the last received SOF packet. The frame number is incremented for every frame sent by the host and it is useful for Isochronous transfers. This bit field is updated on the generation of an SOF interrupt.

USB device address (USB_DADDR)

Address offset: 0x4C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EF	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
Res.								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved

Bit 7 **EF:** *Enable Function*

This bit is set by the software to enable the USB device. The address of this device is contained in the following ADD[6:0] bits. If this bit is at ‘0’ no transactions are handled, irrespective of the settings of USB_EPnR registers.

Bits 6:0 **ADD[6:0]:** *Device Address*

These bits contain the USB function address assigned by the host PC during the enumeration process. Both this field and the Endpoint Address (EA) field in the associated USB_EPnR register must match with the information contained in a USB token in order to handle a transaction to the required endpoint.

Buffer table address (USB_BTABLE)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BTABLE[15:3]													Reserved		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.		

Bits 15:3 **BTABLE[15:3]:** *Buffer Table*.

These bits contain the start address of the buffer allocation table inside the dedicated packet memory. This table describes each endpoint buffer location and size and it must be aligned to an 8 byte boundary (the 3 least significant bits are always ‘0’). At the beginning of every transaction addressed to this device, the USP peripheral reads the element of this table related to the addressed endpoint, to get its buffer start location and the buffer size (Refer to *Structure and usage of packet buffers on page 462*).

Bits 2:0 Reserved, forced by hardware to 0.

20.5.2 Endpoint-specific registers

The number of these registers varies according to the number of endpoints that the USB peripheral is designed to handle. The USB peripheral supports up to 8 bidirectional endpoints. Each USB device must support a control endpoint whose address (EA bits) must be set to 0. The USB peripheral behaves in an undefined way if multiple endpoints are enabled having the same endpoint number value. For each endpoint, an USB_EPnR register is available to store the endpoint specific information.

USB endpoint n register (USB_EPnR), n=[0..7]

Address offset: 0x00 to 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR_ RX	DTOG _RX	STAT_RX[1:0]		SETUP	EP TYPE[1:0]		EP_ KIND	CTR_ TX	DTOG_ TX	STAT_TX[1:0]		EA[3:0]			
rc_w0	t	t	t	r	rw	rw	rw	rc_w0	t	t	t	rw	rw	rw	rw

They are also reset when an USB reset is received from the USB bus or forced through bit FRES in the CTLR register, except the CTR_RX and CTR_TX bits, which are kept unchanged to avoid missing a correct packet notification immediately followed by an USB reset event. Each endpoint has its USB_EPnR register where *n* is the endpoint identifier.

Read-modify-write cycles on these registers should be avoided because between the read and the write operations some bits could be set by the hardware and the next write would modify them before the CPU has the time to detect the change. For this purpose, all bits affected by this problem have an ‘invariant’ value that must be used whenever their modification is not required. It is recommended to modify these registers with a load instruction where all the bits, which can be modified only by the hardware, are written with their ‘invariant’ value.

Bit 15 **CTR_RX:** *Correct Transfer for reception*

This bit is set by the hardware when an OUT/SETUP transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated. The type of occurred transaction, OUT or SETUP, can be determined from the SETUP bit described below.

A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only ‘0’ can be written, writing 1 has no effect.

Bit 14 **DTOG_RX:** *Data Toggle, for reception transfers*

If the endpoint is not Isochronous, this bit contains the expected value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be received. Hardware toggles this bit, when the ACK handshake is sent to the USB host, following a data packet reception having a matching data PID value; if the endpoint is defined as a control one, hardware clears this bit at the reception of a SETUP PID addressed to this endpoint.
If the endpoint is using the double-buffering feature this bit is used to support packet buffer swapping too (Refer to *Section 20.4.3: Double-buffered endpoints*).
If the endpoint is Isochronous, this bit is used only to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to *Section 20.4.4: Isochronous transfers*). Hardware toggles this bit just after the end of data packet reception, since no handshake is used for isochronous transfers.
This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force specific data toggle/packet buffer usage. When the application software writes ‘0’, the value of DTOG_RX remains unchanged, while writing ‘1’ makes the bit value toggle. This bit is read/write but it can be only toggled by writing 1.

Bits 13:12 **STAT_RX [1:0]:** *Status bits, for reception transfers*

These bits contain information about the endpoint status, which are listed in *Table 138: Reception status encoding on page 483*.These bits can be toggled by software to initialize their value. When the application software writes ‘0’, the value remains unchanged, while writing ‘1’ makes the bit value toggle. Hardware sets the STAT_RX bits to NAK when a correct transfer has occurred (CTR_RX=1) corresponding to a OUT or SETUP (control only) transaction addressed to this endpoint, so the software has the time to elaborate the received data before it acknowledge a new transaction
Double-buffered bulk endpoints implement a special transaction flow control, which control the status based upon buffer availability condition (Refer to *Section 20.4.3: Double-buffered endpoints*).
If the endpoint is defined as Isochronous, its status can be only “VALID” or “DISABLED”, so that the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT_RX bits to ‘STALL’ or ‘NAK’ for an Isochronous endpoint, the USB peripheral behavior is not defined. These bits are read/write but they can be only toggled by writing ‘1’.

Bit 11 **SETUP:** *Setup transaction completed*

This bit is read-only and it is set by the hardware when the last completed transaction is a SETUP. This bit changes its value only for control endpoints. It must be examined, in the case of a successful receive transaction (CTR_RX event), to determine the type of transaction occurred. To protect the interrupt service routine from the changes in SETUP bits due to next incoming tokens, this bit is kept frozen while CTR_RX bit is at 1; its state changes when CTR_RX is at 0. This bit is read-only.

Bits 10:9 **EP_TYPE[1:0]:** *Endpoint type*

These bits configure the behavior of this endpoint as described in *Table 139: Endpoint type encoding on page 483*. Endpoint 0 must always be a control endpoint and each USB function must have at least one control endpoint which has address 0, but there may be other control endpoints if required. Only control endpoints handle SETUP transactions, which are ignored by endpoints of other kinds. SETUP transactions cannot be answered with NAK or STALL. If a control endpoint is defined as NAK, the USB peripheral will not answer, simulating a receive error, in the receive direction when a SETUP transaction is received. If the control endpoint is defined as STALL in the receive direction, then the SETUP packet will be accepted anyway, transferring data and issuing the CTR interrupt. The reception of OUT transactions is handled in the normal way, even if the endpoint is a control one.
Bulk and interrupt endpoints have very similar behavior and they differ only in the special feature available using the EP_KIND configuration bit.
The usage of Isochronous endpoints is explained in *Section 20.4.4: Isochronous transfers*

Bit 8 **EP_KIND:** *Endpoint Kind*

The meaning of this bit depends on the endpoint type configured by the EP_TYPE bits. *Table 140* summarizes the different meanings.

DBL_BUF: This bit is set by the software to enable the double-buffering feature for this bulk endpoint. The usage of double-buffered bulk endpoints is explained in *Section 20.4.3: Double-buffered endpoints*.

STATUS_OUT: This bit is set by the software to indicate that a status out transaction is expected: in this case all OUT transactions containing more than zero data bytes are answered 'STALL' instead of 'ACK'. This bit may be used to improve the robustness of the application to protocol errors during control transfers and its usage is intended for control endpoints only. When STATUS_OUT is reset, OUT transactions can have any number of bytes, as required.

Bit 7 **CTR_TX:** *Correct Transfer for transmission*

This bit is set by the hardware when an IN transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in the USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated.

A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only '0' can be written.

Bit 6 **DTOG_TX:** *Data Toggle, for transmission transfers*

If the endpoint is non-isochronous, this bit contains the required value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be transmitted. Hardware toggles this bit when the ACK handshake is received from the USB host, following a data packet transmission. If the endpoint is defined as a control one, hardware sets this bit to 1 at the reception of a SETUP PID addressed to this endpoint.

If the endpoint is using the double buffer feature, this bit is used to support packet buffer swapping too (Refer to *Section 20.4.3: Double-buffered endpoints*)

If the endpoint is Isochronous, this bit is used to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to *Section 20.4.4: Isochronous transfers*). Hardware toggles this bit just after the end of data packet transmission, since no handshake is used for Isochronous transfers.

This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force a specific data toggle/packet buffer usage. When the application software writes '0', the value of DTOG_TX remains unchanged, while writing '1' makes the bit value toggle. This bit is read/write but it can only be toggled by writing 1.

Bits 5:4 **STAT_TX [1:0]:** *Status bits, for transmission transfers*

These bits contain the information about the endpoint status, listed in *Table 141*. These bits can be toggled by the software to initialize their value. When the application software writes '0', the value remains unchanged, while writing '1' makes the bit value toggle. Hardware sets the STAT_TX bits to NAK, when a correct transfer has occurred (CTR_TX=1) corresponding to a IN or SETUP (control only) transaction addressed to this endpoint. It then waits for the software to prepare the next set of data to be transmitted.

Double-buffered bulk endpoints implement a special transaction flow control, which controls the status based on buffer availability condition (Refer to *Section 20.4.3: Double-buffered endpoints*).

If the endpoint is defined as Isochronous, its status can only be "VALID" or "DISABLED". Therefore, the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT_TX bits to 'STALL' or 'NAK' for an Isochronous endpoint, the USB peripheral behavior is not defined. These bits are read/write but they can be only toggled by writing '1'.

Bits 3:0 **EA[3:0]:** *Endpoint Address.*

Software must write in this field the 4-bit address used to identify the transactions directed to this endpoint. A value must be written before enabling the corresponding endpoint.

Table 138. Reception status encoding

STAT_RX[1:0]	Meaning
00	**DISABLED:** all reception requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all reception requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all reception requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for reception.

Table 139. Endpoint type encoding

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO
11	INTERRUPT

Table 140. Endpoint kind meaning

EP_TYPE[1:0]		EP_KIND Meaning
00	BULK	DBL_BUF
01	CONTROL	STATUS_OUT
10	ISO	Not used
11	INTERRUPT	Not used

Table 141. Transmission status encoding

STAT_TX[1:0]	Meaning
00	**DISABLED:** all transmission requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all transmission requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all transmission requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for transmission.

20.5.3 Buffer descriptor table

Although the buffer descriptor table is located inside the packet buffer memory, its entries can be considered as additional registers used to configure the location and size of the packet buffers used to exchange data between the USB macro cell and the STM32F10xxx. Due to the common APB bridge limitation on word addressability, all packet memory locations are accessed by the APB using 32-bit aligned addresses, instead of the actual memory location addresses utilized by the USB peripheral for the USB_BTABLE register

and buffer description table locations.
In the following pages two location addresses are reported: the one to be used by application software while accessing the packet memory, and the local one relative to USB Peripheral access. To obtain the correct STM32F10xxx memory address value to be used in the application software while accessing the packet memory, the actual memory location address must be multiplied by two. The first packet memory location is located at 0x4000 6000. The buffer descriptor table entry associated with the USB_EPnR registers is described below.

A thorough explanation of packet buffers and the buffer descriptor table usage can be found in *Structure and usage of packet buffers on page 462*.

Transmission buffer address n (USB_ADDRn_TX)

Address offset: [USB_BTABLE] + n*16

USB local address: [USB_BTABLE] + n*8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRn_TX[15:1]															-
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Bits 15:1 **ADDRn_TX[15:1]:** *Transmission Buffer Address*

These bits point to the starting address of the packet buffer containing data to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Bit 0 Must always be written as ‘0’ since packet memory is word-wide and all packet buffers must be word-aligned.

Transmission byte count n (USB_COUNTn_TX)

Address offset: [USB_BTABLE] + n*16 + 4

USB local Address: [USB_BTABLE] + n*8 + 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-						COUNTn_TX[9:0]									
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 These bits are not used since packet size is limited by USB specifications to 1023 bytes. Their value is not considered by the USB peripheral.

Bits 9:0 **COUNTn_TX[9:0]:** *Transmission Byte Count*

These bits contain the number of bytes to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Note: Double-buffered and Isochronous IN Endpoints have two USB_COUNTn_TX registers: named USB_COUNTn_TX_1 and USB_COUNTn_TX_0 with the following content.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-						COUNTn_TX_1[9:0]									
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-						COUNTn_TX_0[9:0]									
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Reception buffer address n (USB_ADDRn_RX)

Address offset: [USB_BTABLE] + n*16 + 8

USB local Address: [USB_BTABLE] + n*8 + 4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRn_RX[15:1]															-
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Bits 15:1 **ADDRn_RX[15:1]:** *Reception Buffer Address*

These bits point to the starting address of the packet buffer, which will contain the data received by the endpoint associated with the USB_EPnR register at the next OUT/SETUP token addressed to it.

Bit 0 This bit must always be written as ‘0’ since packet memory is word-wide and all packet buffers must be word-aligned.

Reception byte count n (USB_COUNTn_RX)

Address offset: [USB_BTABLE] + n*16 + 12

USB local Address: [USB_BTABLE] + n*8 + 6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLSIZE	NUM_BLOCK[4:0]					COUNTn_RX[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r

This table location is used to store two different values, both required during packet reception. The most significant bits contains the definition of allocated buffer size, to allow buffer overflow detection, while the least significant part of this location is written back by the USB peripheral at the end of reception to give the actual number of received bytes. Due to the restrictions on the number of available bits, buffer size is represented using the number of allocated memory blocks, where block size can be selected to choose the trade-off between fine-granularity/small-buffer and coarse-granularity/large-buffer. The size of allocated buffer is a part of the endpoint descriptor and it is normally defined during the enumeration process according to its maxPacketSize parameter value (See “Universal Serial Bus Specification”).

Bit 15 **BL_SIZE:** *BLock SIZE.*

This bit selects the size of memory block used to define the allocated buffer area.

– If BL_SIZE=0, the memory block is 2 byte large, which is the minimum block allowed in a word-wide memory. With this block size the allocated buffer size ranges from 2 to 62 bytes.

– If BL_SIZE=1, the memory block is 32 `byte` large, which allows to reach the maximum packet length defined by USB specifications. With this block size the allocated buffer size ranges from 32 to 1024 bytes, which is the longest packet size allowed by USB standard specifications.

Bits 14:10 **NUM_BLOCK[4:0]:** *Number of blocks.*

These bits define the number of memory blocks allocated to this packet buffer. The actual amount of allocated memory depends on the BL_SIZE value as illustrated in *Table 142*.

Bits 9:0 **COUNTn_RX[9:0]:** *Reception Byte Count*

These bits contain the number of bytes received by the endpoint associated with the USB_EPnR register during the last OUT/SETUP transaction addressed to it.

Note: Double-buffered and Isochronous IN Endpoints have two USB_COUNTn_TX registers: named USB_COUNTn_TX_1 and USB_COUNTn_TX_0 with the following content.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BLSIZE_1	NUM_BLOCK_1[4:0]					COUNTn_RX_1[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLSIZE_0	NUM_BLOCK_0[4:0]					COUNTn_RX_0[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r

Table 142. Definition of allocated buffer memory

Value of NUM_BLOCK[4:0]	Memory allocated when BL_SIZE=0	Memory allocated when BL_SIZE=1
0 (‘00000’)	Not allowed	32 bytes
1 (‘00001’)	2 bytes	64 bytes
2 (‘00010’)	4 bytes	96 bytes
3 (‘00011’)	6 bytes	128 bytes
...	...	...
15 (‘01111’)	30 bytes	512 bytes
16 (‘10000’)	32 bytes	N/A
17 (‘10001’)	34 bytes	N/A
18 (‘10010’)	36 bytes	N/A
...	...	...
30 (‘11110’)	60 bytes	N/A
31 (‘11111’)	62 bytes	N/A

20.5.4 USB register map

The table below provides the USB register map and reset values.

Table 143. USB register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**USB_EP0R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	**USB_EP1R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**USB_EP2R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**USB_EP3R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**USB_EP4R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**USB_EP5R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**USB_EP6R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**USB_EP7R**	Reserved																CTR_RX	DTOG_RX	STAT_RX [1:0]		SETUP	EP TYPE [1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX [1:0]		EA[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20-0x3F	Reserved																																
0x40	**USB_CNTR**	Reserved																CTRM	PMAOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LPMODE	PDWN	FRES
	Reset value																	0	0	0	0	0	0	0	0				0	0	0	1	1
0x44	**USB_ISTR**	Reserved																CTR	PMAOVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved			DIR	EP_ID[3:0]			
	Reset value																	0	0	0	0	0	0	0	0				0	0	0	0	0
0x48	**USB_FNR**	Reserved																RXDP	RXDM	LCK	LSOF [1:0]		FN[10:0]										
	Reset value																	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x
0x4C	**USB_DADDR**	Reserved																Reserved								EF	ADD[6:0]						
	Reset value																									0	0	0	0	0	0	0	0

Table 143. USB register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x50	**USB_BTABLE**	Reserved																BTABLE[15:3]													Reserved		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0			

Note: Refer to Table 1 on page 35 for the register boundary addresses.

21 Controller area network (bxCAN)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the STM32F103xx performance line only.

21.1 bxCAN introduction

The **Basic Extended CAN** peripheral, named **bxCAN**, interfaces the CAN network. It supports the CAN protocols version 2.0A and B. It has been designed to manage a high number of incoming messages efficiently with a minimum CPU load. It also meets the priority requirements for transmit messages.

For safety-critical applications, the CAN controller provides all hardware functions for supporting the CAN Time Triggered Communication option.

21.2 bxCAN main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

Reception

- Two receive FIFOs with three stages
- 14 scalable filter banks
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

Time-triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Time Stamp sent in last two data bytes

Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

Note: In Medium-density and High-density devices the USB and CAN share a dedicated 512-byte SRAM memory for data transmission and reception, and so they cannot be used concurrently (the shared SRAM is accessed through CAN and USB exclusively). The USB and CAN can be used in the same application but not at the same time.

21.2.1 General description

In today's CAN applications, the number of nodes in a network is increasing and often several networks are linked together via gateways. Typically the number of messages in the system (and thus to be handled by each node) has significantly increased. In addition to the application messages, Network Management and Diagnostic messages have been introduced.

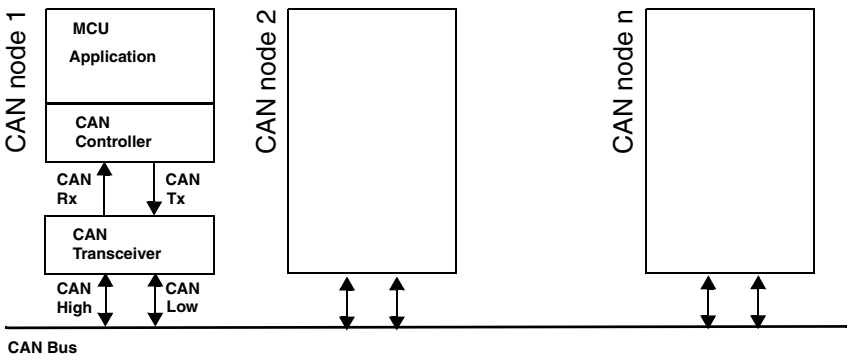
- An enhanced filtering mechanism is required to handle each type of message.

Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception have to be reduced.

- A receive FIFO scheme allows the CPU to be dedicated to application tasks for a long time period without losing messages.

The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.

Figure 190. CAN network topology



CAN 2.0B active core

The bxCAN module handles the transmission and the reception of CAN messages fully autonomously. Standard identifiers (11-bit) and extended identifiers (29-bit) are fully supported by hardware.

Control, status and configuration registers

The application uses these registers to:

- Configure CAN parameters, e.g. baud rate
- Request transmissions
- Handle receptions
- Manage interrupts
- Get diagnostic information

Tx mailboxes

Three transmit mailboxes are provided to the software for setting up messages. The transmission Scheduler decides which mailbox has to be transmitted first.

Acceptance filters

The bxCAN provides 14 scalable/configurable identifier filter banks for selecting the incoming messages the software needs and discarding the others.

Receive FIFO

Two receive FIFOs are used by hardware to store the incoming messages. Three complete messages can be stored in each FIFO. The FIFOs are managed completely by hardware.

Figure 191. CAN general block diagram

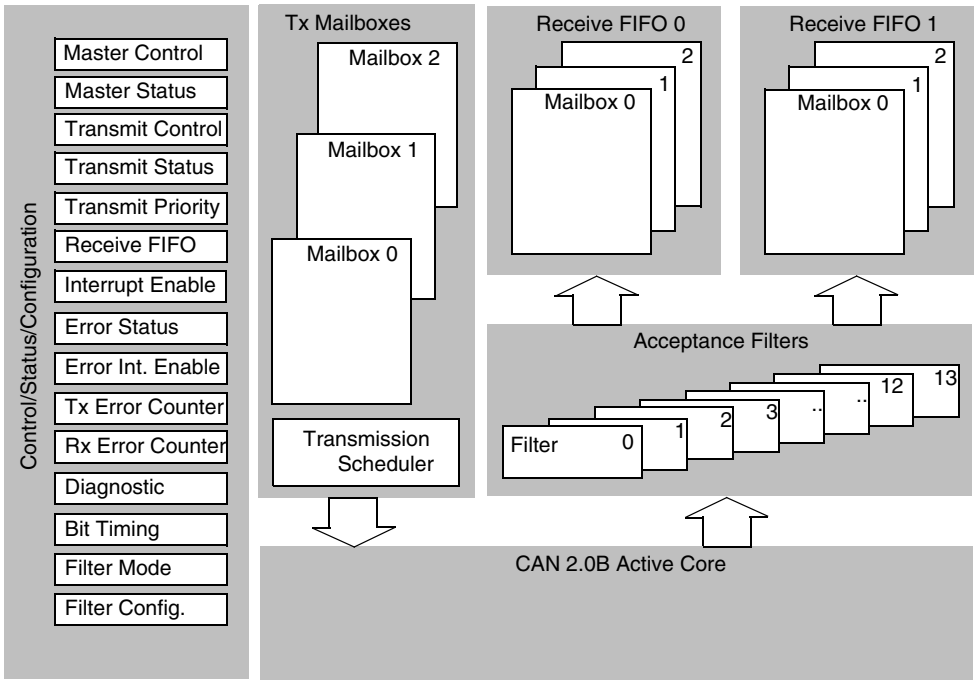
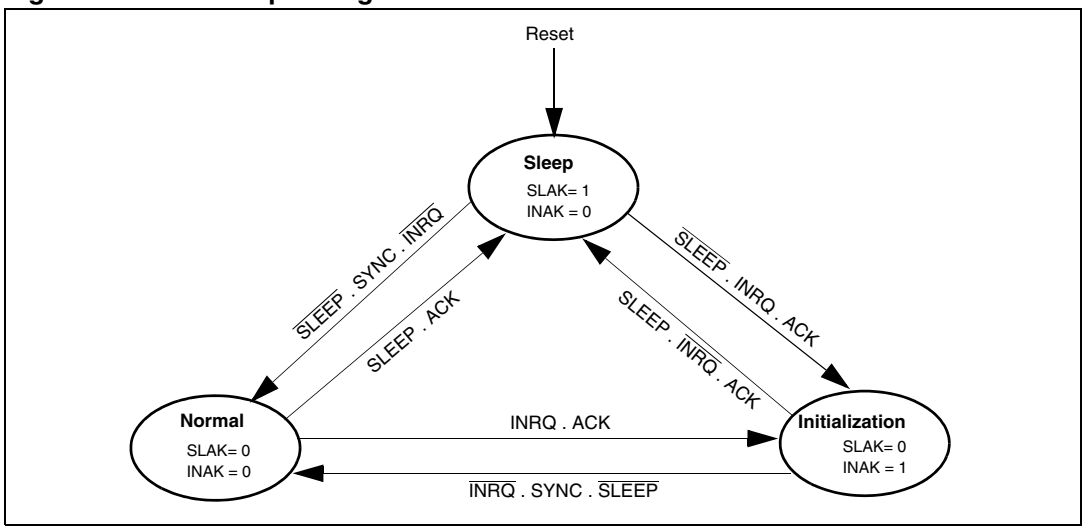


Figure 192.

Figure 193. bxCAN operating modes



Note: 1 ACK = The wait state during which hardware confirms a request by setting the INAK or SLAK bits in the CAN_MSR register

2 SYNC = The state during which bxCAN waits until the CAN bus is idle, meaning 11 consecutive recessive bits have been monitored on CANRX

21.3 bxCAN operating modes

bxCAN has three main operating modes: **initialization**, **normal** and **Sleep**. After a hardware reset, bxCAN is in Sleep mode to reduce power consumption and an internal pull-up is active on CANTX. The software requests bxCAN to enter **initialization** or **Sleep** mode by setting the INRQ or SLEEP bits in the CAN_MCR register. Once the mode has been entered, bxCAN confirms it by setting the INAK or SLAK bits in the CAN_MSR register and the internal pull-up is disabled. When neither INAK nor SLAK are set, bxCAN is in **normal** mode. Before entering **normal** mode bxCAN always has to **synchronize** on the CAN bus. To synchronize, bxCAN waits until the CAN bus is idle, this means 11 consecutive recessive bits have been monitored on CANRX.

21.3.1 Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter this mode the software sets the INRQ bit in the CAN_MCR register and waits until the hardware has confirmed the request by setting the INAK bit in the CAN_MSR register.

To leave Initialization mode, the software clears the INQR bit. bxCAN has left Initialization mode once the INAK bit has been cleared by hardware.

While in Initialization Mode, all message transfers to and from the CAN bus are stopped and the status of the CAN bus output CANTX is recessive (high).

Entering Initialization Mode does not change any of the configuration registers.

To initialize the CAN Controller, software has to set up the Bit Timing (CAN_BTR) and CAN options (CAN_MCR) registers.

To initialize the registers associated with the CAN filter banks (mode, scale, FIFO assignment, activation and filter values), software has to set the FINIT bit (CAN_FMR). Filter initialization also can be done outside the initialization mode.

Note: When FINIT=1, CAN reception is deactivated.

The filter values also can be modified by deactivating the associated filter activation bits (in the CAN_FA1R register).

If a filter bank is not used, it is recommended to leave it non active (leave the corresponding FACT bit cleared).

21.3.2 Normal mode

Once the initialization has been done, the software must request the hardware to enter Normal mode, to synchronize on the CAN bus and start reception and transmission. Entering Normal mode is done by clearing the INRQ bit in the CAN_MCR register and waiting until the hardware has confirmed the request by clearing the INAK bit in the CAN_MSR register. Afterwards, the bxCAN synchronizes with the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (≡ Bus Idle) before it can take part in bus activities and start message transfer.

The initialization of the filter values is independent from Initialization Mode but must be done while the filter is not active (corresponding FACTx bit cleared). The filter scale and mode configuration must be configured before entering Normal Mode.

21.3.3 Sleep mode (low power)

To reduce power consumption, bxCAN has a low-power mode called Sleep mode. This mode is entered on software request by setting the SLEEP bit in the CAN_MCR register. In this mode, the bxCAN clock is stopped, however software can still access the bxCAN mailboxes.

If software requests entry to **initialization** mode by setting the INRQ bit while bxCAN is in **Sleep** mode, it must also clear the SLEEP bit.

bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

On CAN bus activity detection, hardware automatically performs the wakeup sequence by clearing the SLEEP bit if the AWUM bit in the CAN_MCR register is set. If the AWUM bit is cleared, software has to clear the SLEEP bit when a wakeup interrupt occurs, in order to exit from Sleep mode.

Note: If the wakeup interrupt is enabled (WKUIE bit set in CAN_IER register) a wakeup interrupt will be generated on detection of CAN bus activity, even if the bxCAN automatically performs the wakeup sequence.

After the SLEEP bit has been cleared, Sleep mode is exited once bxCAN has synchronized with the CAN bus, refer to *Figure 193: bxCAN operating modes*. The Sleep mode is exited once the SLAK bit has been cleared by hardware.

21.3.4 Test mode

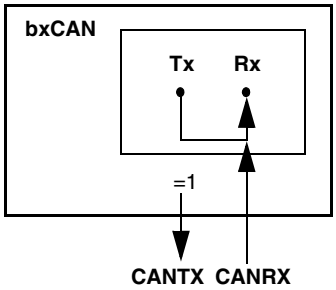
Test mode can be selected by the SILM and LBKM bits in the CAN_BTR register. These bits must be configured while bxCAN is in Initialization mode. Once test mode has been selected, the INRQ bit in the CAN_MCR register must be reset to enter Normal mode.

21.3.5 Silent mode

The bxCAN can be put in Silent mode by setting the SILM bit in the CAN_BTR register.

In Silent mode, the bxCAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the bxCAN has to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).

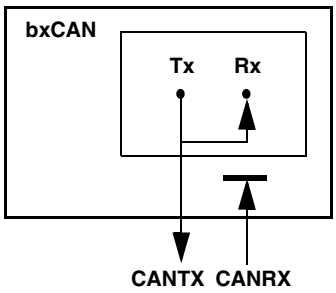
Figure 194. bxCAN in silent mode



21.3.6 Loop back mode

The bxCAN can be set in Loop Back Mode by setting the LBKM bit in the CAN_BTR register. In Loop Back Mode, the bxCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in a Receive mailbox.

Figure 195. bxCAN in loop back mode



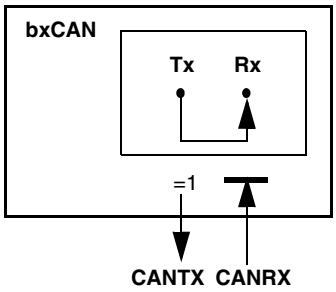
This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

21.3.7 Loop back combined with silent mode

It is also possible to combine Loop Back mode and Silent mode by setting the LBKM and SILM bits in the CAN_BTR register. This mode can be used for a “Hot Selftest”, meaning the bxCAN can be tested like in Loop Back mode but without affecting a running CAN system

connected to the CANTX and CANRX pins. In this mode, the CANRX pin is disconnected from the bxCAN and the CANTX pin is held recessive.

Figure 196. bxCAN in combined mode



21.4 bxCAN functional description

21.4.1 Transmission handling

In order to transmit a message, the application must select one **empty** transmit mailbox, set up the identifier, the data length code (DLC) and the data before requesting the transmission by setting the corresponding TXRQ bit in the CAN_TIxR register. Once the mailbox has left **empty** state, the software no longer has write access to the mailbox registers. Immediately after the TXRQ bit has been set, the mailbox enters **pending** state and waits to become the highest priority mailbox, see *Transmit Priority*. As soon as the mailbox has the highest priority it will be **scheduled** for transmission. The transmission of the message of the scheduled mailbox will start (enter **transmit** state) when the CAN bus becomes idle. Once the mailbox has been successfully transmitted, it will become **empty** again. The hardware indicates a successful transmission by setting the RQCP and TXOK bits in the CAN_TSR register.

If the transmission fails, the cause is indicated by the ALST bit in the CAN_TSR register in case of an Arbitration Lost, and/or the TERR bit, in case of transmission error detection.

Transmit priority

By identifier:

When more than one transmit mailbox is pending, the transmission order is given by the identifier of the message stored in the mailbox. The message with the lowest identifier value has the highest priority according to the arbitration of the CAN protocol. If the identifier values are equal, the lower mailbox number will be scheduled first.

By transmit request order:

The transmit mailboxes can be configured as a transmit FIFO by setting the TXFP bit in the CAN_MCR register. In this mode the priority order is given by the transmit request order.

This mode is very useful for segmented transmission.

Abort

A transmission request can be aborted by the user setting the ABRQ bit in the CAN_TSR register. In **pending** or **scheduled** state, the mailbox is aborted immediately. An abort

request while the mailbox is in **transmit** state can have two results. If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN_TSR register. If the transmission fails, the mailbox becomes **scheduled,** the transmission is aborted and becomes **empty** with TXOK cleared. In all cases the mailbox will become **empty** again at least at the end of the current transmission.

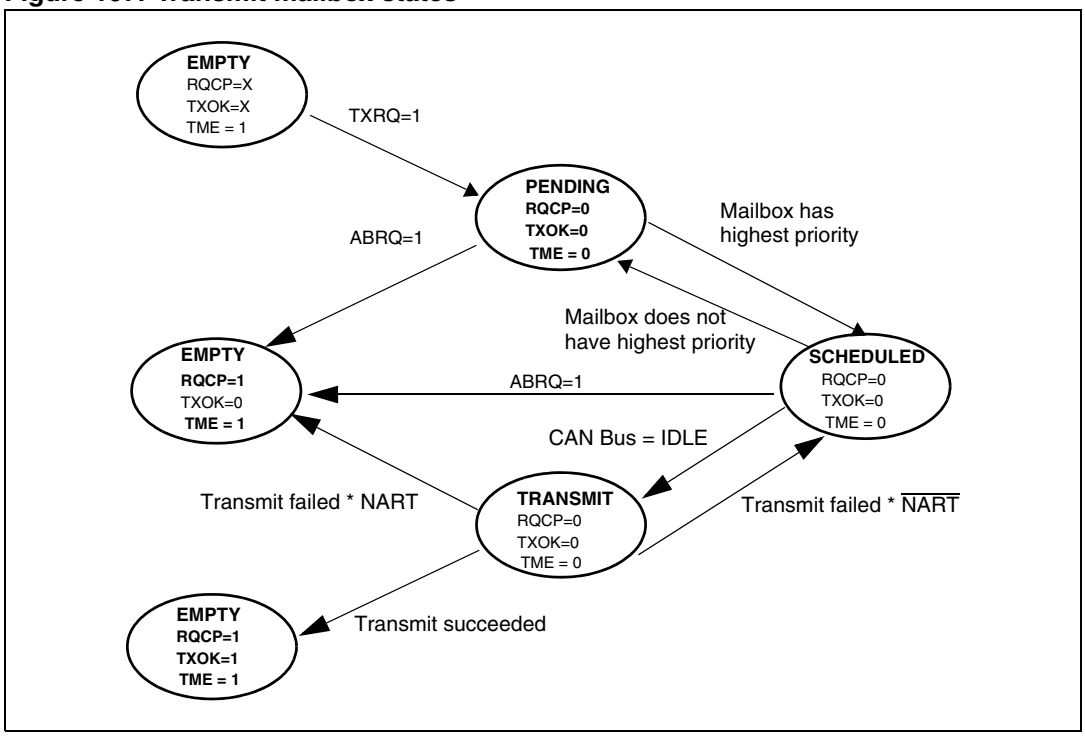
Non-automatic retransmission mode

This mode has been implemented in order to fulfil the requirement of the Time Triggered Communication option of the CAN standard. To configure the hardware in this mode the NART bit in the CAN_MCR register must be set.

In this mode, each transmission is started only once. If the first attempt fails, due to an arbitration loss or an error, the hardware will not automatically restart the message transmission.

At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP bit in the CAN_TSR register. The result of the transmission is indicated in the CAN_TSR register by the TXOK, ALST and TERR bits.

Figure 197. Transmit mailbox states



21.4.2 Time triggered communication mode

In this mode, the internal counter of the CAN hardware is activated and used to generate the Time Stamp value stored in the CAN_RDTxR/CAN_TDTxR registers, respectively (for Rx and Tx mailboxes). The internal counter is incremented each CAN bit time (refer to *Section 21.4.7: Bit timing*). The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.

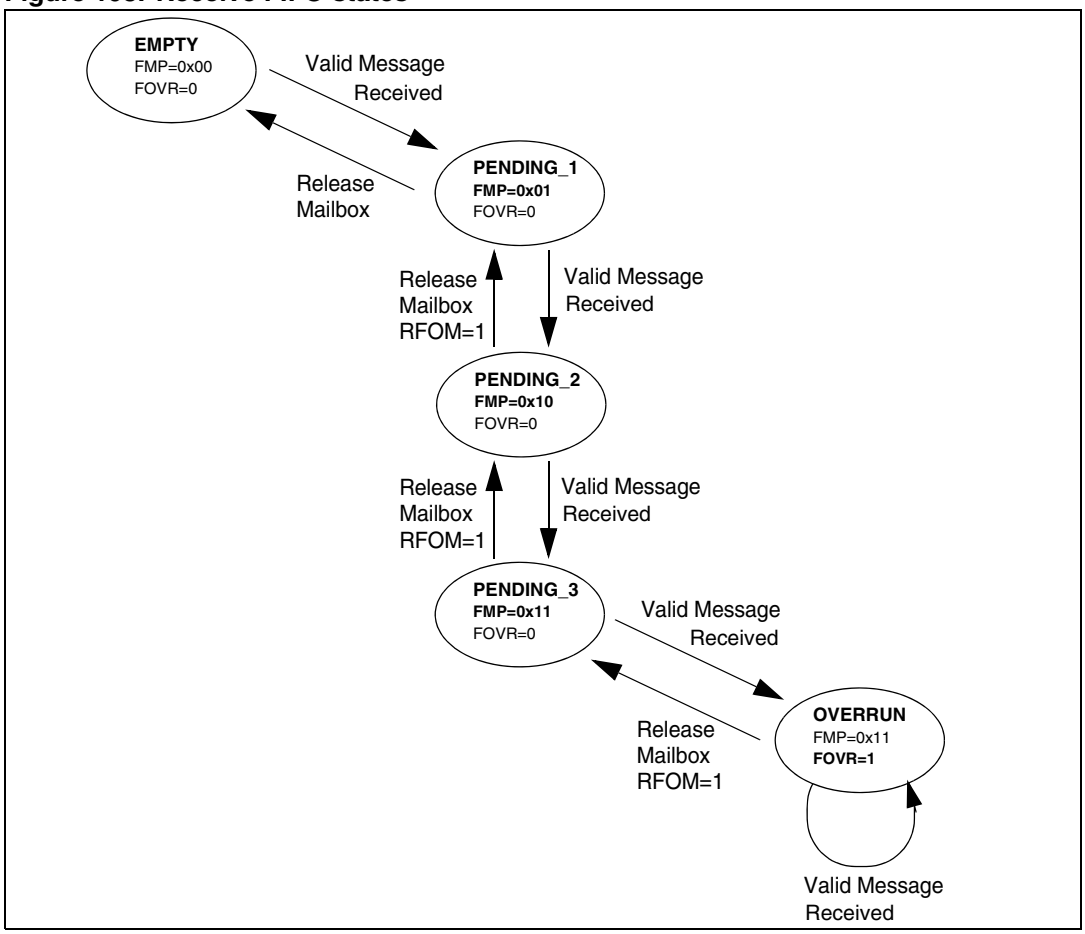
21.4.3 Reception handling

For the reception of CAN messages, three mailboxes organized as a FIFO are provided. In order to save CPU load, simplify the software and guarantee data consistency, the FIFO is managed completely by hardware. The application accesses the messages stored in the FIFO through the FIFO output mailbox.

Valid message

A received message is considered as valid **when** it has been received correctly according to the CAN protocol (no error until the last but one bit of the EOF field) **and** It passed through the identifier filtering successfully, see *Section 21.4.4: Identifier filtering*.

Figure 198. Receive FIFO states



FIFO management

Starting from the **empty** state, the first valid message received is stored in the FIFO which becomes **pending_1**. The hardware signals the event setting the FMP[1:0] bits in the CAN_RFR register to the value 01b. The message is available in the FIFO output mailbox. The software reads out the mailbox content and releases it by setting the RFOM bit in the CAN_RFR register. The FIFO becomes **empty** again. If a new valid message has been received in the meantime, the FIFO stays in **pending_1** state and the new message is available in the output mailbox.

If the application does not release the mailbox, the next valid message will be stored in the FIFO which enters **pending_2** state (FMP[1:0] = 10b). The storage process is repeated for the next valid message putting the FIFO into **pending_3** state (FMP[1:0] = 11b). At this point, the software must release the output mailbox by setting the RFOM bit, so that a mailbox is free to store the next valid message. Otherwise the next valid message received will cause a loss of message.

Refer also to *Section 21.4.5: Message storage*

Overrun

Once the FIFO is in **pending_3** state (i.e. the three mailboxes are full) the next valid message reception will lead to an **overrun** and a message will be lost. The hardware signals the overrun condition by setting the FOVR bit in the CAN_RFR register. Which message is lost depends on the configuration of the FIFO:

- If the FIFO lock function is disabled (RFLM bit in the CAN_MCR register cleared) the last message stored in the FIFO will be overwritten by the new incoming message. In this case the latest messages will be always available to the application.
- If the FIFO lock function is enabled (RFLM bit in the CAN_MCR register set) the most recent message will be discarded and the software will have the three oldest messages in the FIFO available.

Reception related interrupts

Once a message has been stored in the FIFO, the FMP[1:0] bits are updated and an interrupt request is generated if the FMPIE bit in the CAN_IER register is set.

When the FIFO becomes full (i.e. a third message is stored) the FULL bit in the CAN_RFR register is set and an interrupt is generated if the FFIE bit in the CAN_IER register is set.

On overrun condition, the FOVR bit is set and an interrupt is generated if the FOVIE bit in the CAN_IER register is set.

21.4.4 Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.

To fulfil this requirement, the bxCAN Controller provides 14 configurable and scalable filter banks (13-0) to the application. in order to receive only the messages the software needs.This hardware filtering saves CPU resources which would be otherwise needed to perform filtering by software. Each filter bank x consists of two 32-bit registers, CAN_FxR0 and CAN_FxR1.

Scalable width

To optimize and adapt the filters to the application needs, each filter bank can be scaled independently. Depending on the filter scale a filter bank provides:

- One 32-bit filter for the STDID[10:0], EXTID[17:0], IDE and RTR bits.
- Two 16-bit filters for the STDID[10:0], RTR, IDE and EXTID[17:15] bits.

Refer to *Figure 199*.

Furthermore, the filters can be configured in mask mode or in identifier list mode.

Mask mode

In **mask** mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.

Identifier list mode

In **identifier list** mode, the mask registers are used as identifier registers. Thus instead of defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.

Filter bank scale and mode configuration

The filter banks are configured by means of the corresponding CAN_FMR register. To configure a filter bank it must be deactivated by clearing the FACT bit in the CAN_FAR register. The filter scale is configured by means of the corresponding FSCx bit in the CAN_FS1R register, refer to *Figure 199*. The **identifier list** or **identifier mask** mode for the corresponding Mask/Identifier registers is configured by means of the FBMx bits in the CAN_FMR register.

To filter a group of identifiers, configure the Mask/Identifier registers in mask mode.

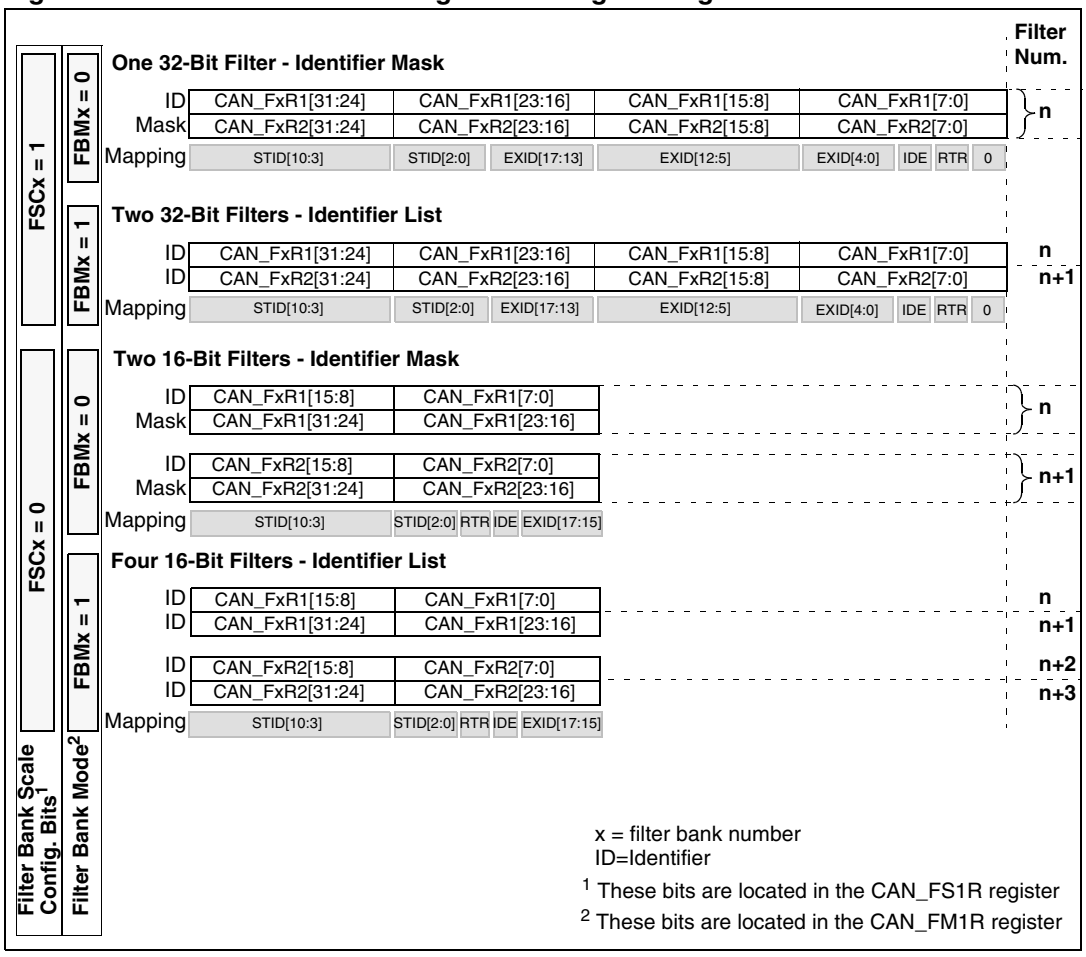
To select single identifiers, configure the Mask/Identifier registers in identifier list mode.

Filters not used by the application should be left deactivated.

Each filter within a filter bank is numbered (called the *Filter Number*) from 0 to a maximum dependent on the mode and the scale of each of the filter banks.

Concerning the filter configuration, refer to *Figure 199*.

Figure 199. Filter bank scale configuration - register organization



Filter match index

Once a message has been received in the FIFO it is available to the application. Typically, application data is copied into SRAM locations. To copy the data to the right location the application has to identify the data by means of the identifier. To avoid this, and to ease the access to the SRAM locations, the CAN controller provides a Filter Match Index.

This index is stored in the mailbox together with the message according to the filter priority rules. Thus each received message has its associated filter match index.

The Filter Match index can be used in two ways:

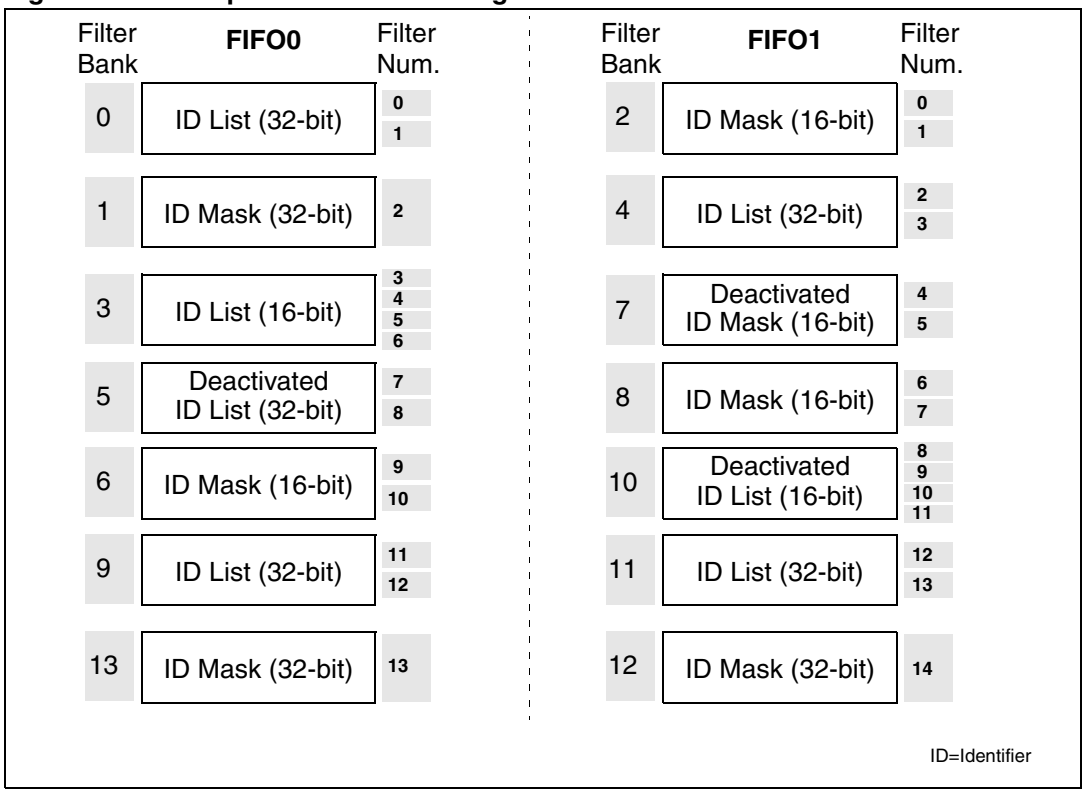
- Compare the Filter Match index with a list of expected values.
- Use the Filter Match Index as an index on an array to access the data destination location.

For non-masked filters, the software no longer has to compare the identifier.

If the filter is masked the software reduces the comparison to the masked bits only.

The index value of the filter number does not take into account the activation state of the filter banks. In addition, two independent numbering schemes are used, one for each FIFO. Refer to *Figure 200* for an example.

Figure 200. Example of filter numbering

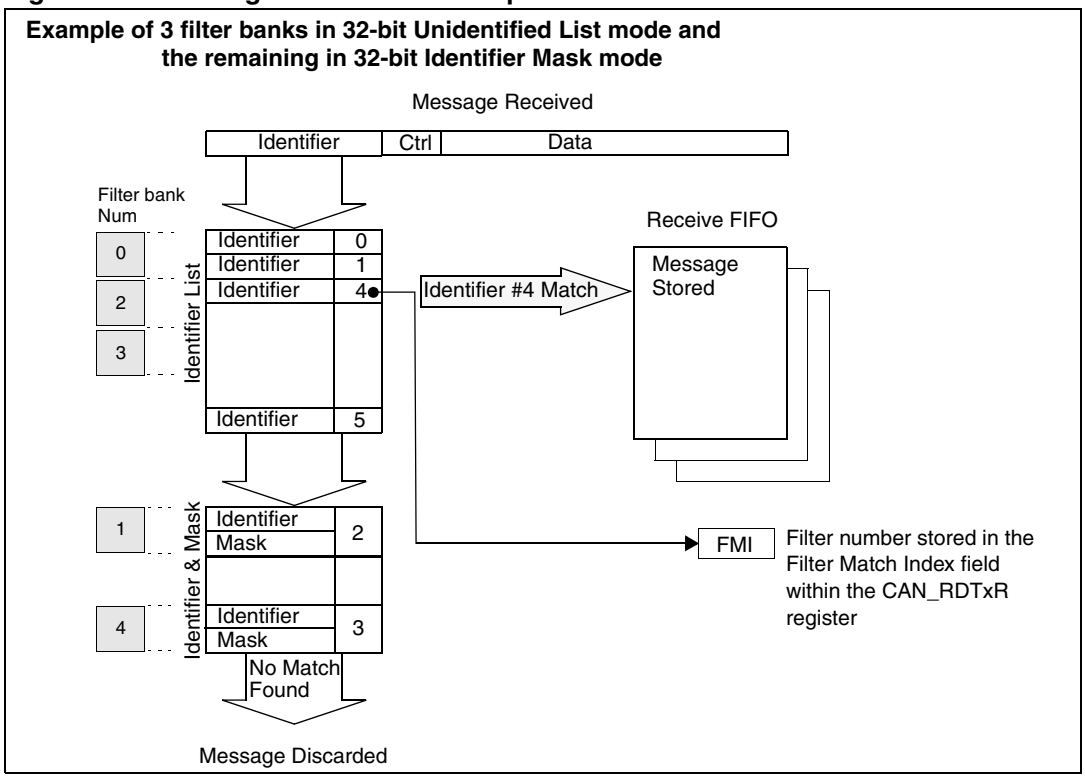


Filter priority rules

Depending on the filter combination it may occur that an identifier passes successfully through several filters. In this case the filter match value stored in the receive mailbox is chosen according to the following priority rules:

- A 32-bit filter takes priority over a 16-bit filter.
- For filters of equal scale, priority is given to the Identifier List mode over the Identifier Mask mode
- For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority).

Figure 201. Filtering mechanism - example



The example above shows the filtering principle of the bxCAN. On reception of a message, the identifier is compared first with the filters configured in identifier list mode. If there is a match, the message is stored in the associated FIFO and the index of the matching filter is stored in the Filter Match Index. As shown in the example, the identifier matches with Identifier #2 thus the message content and FMI 2 is stored in the FIFO.

If there is no match, the incoming identifier is then compared with the filters configured in mask mode.

If the identifier does not match any of the identifiers configured in the filters, the message is discarded by hardware without disturbing the software.

21.4.5 Message storage

The interface between the software and the hardware for the CAN messages is implemented by means of mailboxes. A mailbox contains all information related to a message; identifier, data, control, status and time stamp information.

Transmit mailbox

The software sets up the message to be transmitted in an empty transmit mailbox. The status of the transmission is indicated by hardware in the CAN_TSR register.

Table 144. Transmit mailbox mapping

Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDHxR

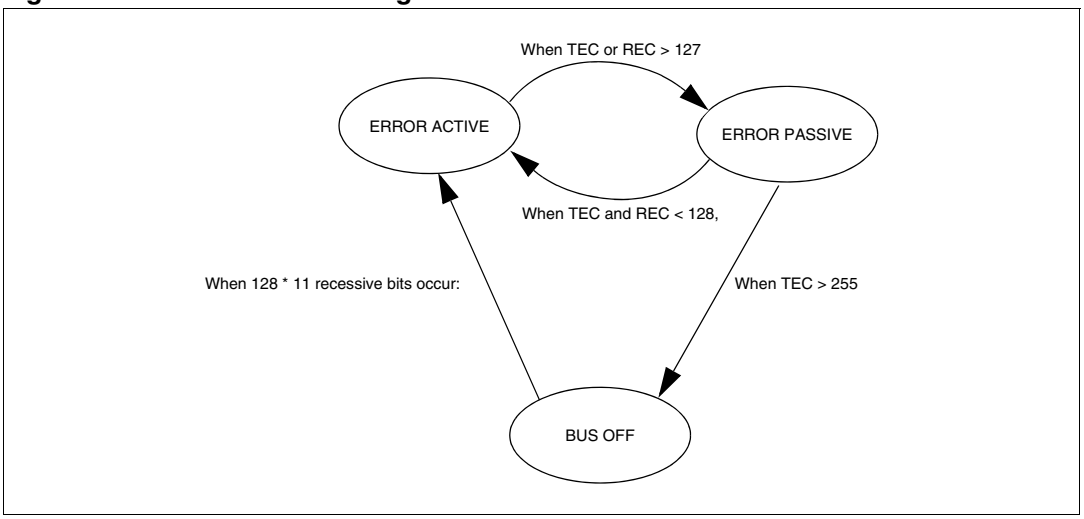
Receive mailbox

When a message has been received, it is available to the software in the FIFO output mailbox. Once the software has handled the message (e.g. read it) the software must release the FIFO output mailbox by means of the RFOM bit in the CAN_RFR register to make the next incoming message available. The filter match index is stored in the MFMI field of the CAN_RDTxR register. The 16-bit time stamp value is stored in the TIME[15:0] field of CAN_RDTxR.

Table 145. Receive mailbox mapping

Offset to receive mailbox base address (bytes)	Register name
0	CAN_RIxR
4	CAN_RDTxR
8	CAN_RDLxR
12	CAN_RDHxR

Figure 202. CAN error state diagram



21.4.6 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (TEC value, in CAN_ESR register) and a Receive Error Counter (REC value, in the CAN_ESR register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, please refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CAN_ESR register. By means of the CAN_IER register (ERRIE bit, etc.), the software can configure the interrupt generation on error detection in a very flexible way.

Bus-Off recovery

The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF bit in CAN_ESR register. In Bus-Off state, the bxCAN is no longer able to transmit and receive messages.

Depending on the ABOM bit in the CAN_MCR register bxCAN will recover from Bus-Off (become error active again) either automatically or on software request. But in both cases the bxCAN has to wait at least for the recovery sequence specified in the CAN standard (128 occurrences of 11 consecutive recessive bits monitored on CANRX).

If ABOM is set, the bxCAN will start the recovering sequence automatically after it has entered Bus-Off state.

If ABOM is cleared, the software must initiate the recovering sequence by requesting bxCAN to enter and to leave initialization mode.

Note: In initialization mode, bxCAN does not monitor the CANRX signal, therefore it cannot complete the recovery sequence. ***To recover, bxCAN must be in normal mode****.*

21.4.7 Bit timing

The bit timing logic monitors the serial bus-line and performs sampling and adjustment of the sample point by synchronizing on the start-bit edge and resynchronizing on the following edges.

Its operation may be explained simply by splitting nominal bit time into three segments as follows:

- **Synchronization segment (SYNC_SEG)**: a bit change is expected to occur within this time segment. It has a fixed length of one time quantum (1 x t_{CAN}).
- **Bit segment 1 (BS1)**: defines the location of the sample point. It includes the PROP_SEG and PHASE_SEG1 of the CAN standard. Its duration is programmable between 1 and 16 time quanta but may be automatically lengthened to compensate for positive phase drifts due to differences in the frequency of the various nodes of the network.
- **Bit segment 2 (BS2)**: defines the location of the transmit point. It represents the PHASE_SEG2 of the CAN standard. Its duration is programmable between 1 and 8 time quanta but may also be automatically shortened to compensate for negative phase drifts.

The resynchronization Jump Width (SJW) defines an upper bound to the amount of lengthening or shortening of the bit segments. It is programmable between 1 and 4 time quanta.

A valid edge is defined as the first transition in a bit time from dominant to recessive bus level provided the controller itself does not send a recessive bit.

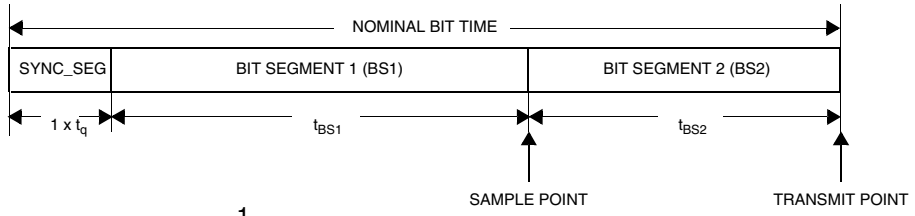
If a valid edge is detected in BS1 instead of SYNC_SEG, BS1 is extended by up to SJW so that the sample point is delayed.

Conversely, if a valid edge is detected in BS2 instead of SYNC_SEG, BS2 is shortened by up to SJW so that the transmit point is moved earlier.

As a safeguard against programming errors, the configuration of the Bit Timing Register (CAN_BTR) is only possible while the device is in Standby mode.

Note: For a detailed description of the CAN bit timing and resynchronization mechanism, please refer to the ISO 11898 standard.

Figure 203. Bit timing



$$\text{BaudRate} = \frac{1}{\text{NominalBitTime}}$$

$$\text{NominalBitTime} = 1 \times t_q + t_{BS1} + t_{BS2}$$

with:

$t_{BS1} = t_q$ x (TS1[3:0] + 1),

$t_{BS2} = t_q$ x (TS2[2:0] + 1),

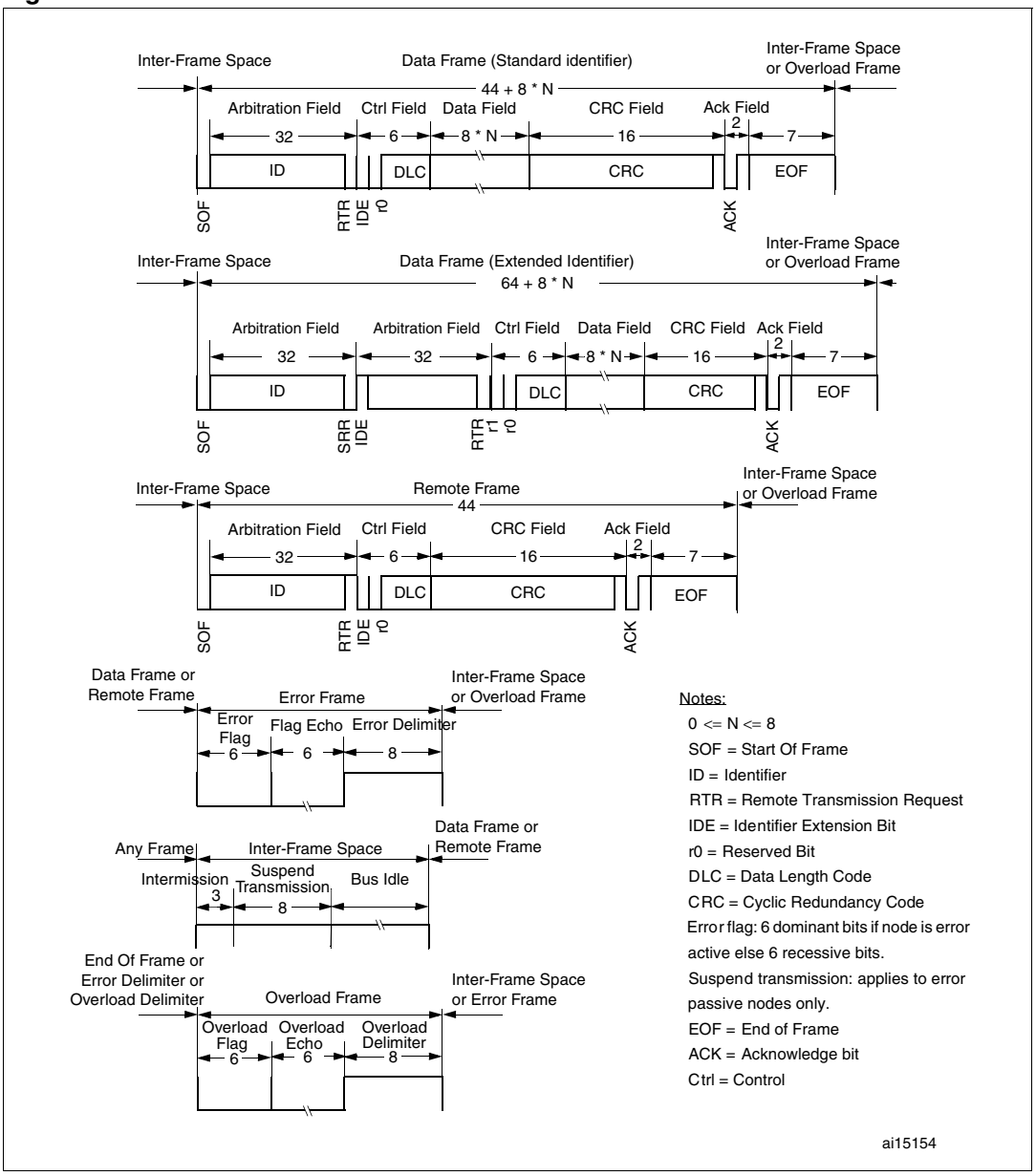
t_q = (BRP[9:0] + 1) x t_{PCLK}

where t_q refers to the Time quantum

t_{PCLK} = time period of the APB clock,

BRP[9:0], TS1[3:0] and TS2[2:0] are defined in the CAN_BTR Register.

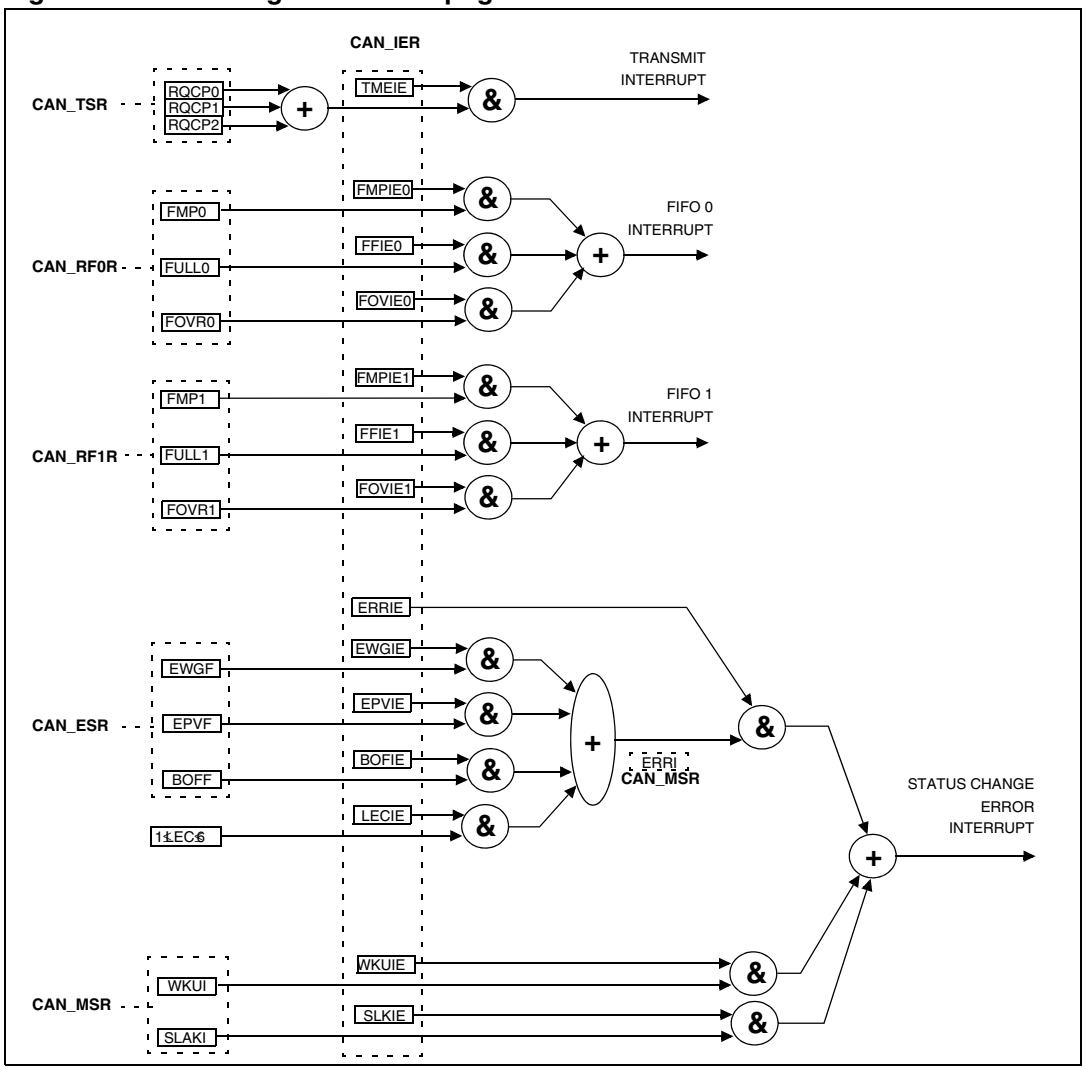
Figure 204. CAN frames



21.5 bxCAN interrupts

Four interrupt vectors are dedicated to bxCAN. Each interrupt source can be independently enabled or disabled by means of the CAN Interrupt Enable Register (CAN_IER).

Figure 205. Event flags and interrupt generation



- The **transmit interrupt** can be generated by the following events:
  - Transmit mailbox 0 becomes empty, RQCP0 bit in the CAN_TSR register set.
  - Transmit mailbox 1 becomes empty, RQCP1 bit in the CAN_TSR register set.
  - Transmit mailbox 2 becomes empty, RQCP2 bit in the CAN_TSR register set.
- The **FIFO 0 interrupt** can be generated by the following events:
  - Reception of a new message, FMP0 bits in the CAN_RF0R register are not ‘00’.
  - FIFO0 full condition, FULL0 bit in the CAN_RF0R register set.
  - FIFO0 overrun condition, FOVR0 bit in the CAN_RF0R register set.
- The **FIFO 1 interrupt** can be generated by the following events:
  - Reception of a new message, FMP1 bits in the CAN_RF1R register are not ‘00’.
  - FIFO1 full condition, FULL1 bit in the CAN_RF1R register set.
  - FIFO1 overrun condition, FOVR1 bit in the CAN_RF1R register set.

- The **error and status change interrupt** can be generated by the following events:
  - Error condition, for more details on error conditions please refer to the CAN Error Status register (CAN_ESR).
  - Wakeup condition, SOF monitored on the CAN Rx signal.
  - Entry into Sleep mode.

21.6 CAN registers

21.6.1 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore the CAN_BTR register can be modified by software only while the CAN hardware is in initialization mode.

Although the transmission of incorrect data will not cause problems at the CAN network level, it can severely disturb the application. A transmit mailbox can be only modified by software while it is in empty state, refer to *Figure 197: Transmit mailbox states*.

The filter values can be modified either deactivating the associated filter banks or by setting the FINIT bit. Moreover, the modification of the filter configuration (scale, mode and FIFO assignment) in CAN_FMxR, CAN_FSxR and CAN_FFAR registers can only be done when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

21.6.2 CAN control and status registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

CAN master control register (CAN_MCR)

Address offset: 0x00
Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Reserved							TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
rs								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bit 15 **RESET:** *bxCAN software master reset*

0: Normal operation.
1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, forced by hardware to 0.

Bit 7 **TTCM**: *Time Triggered Communication Mode*

0: Time Triggered Communication mode disabled.
1: Time Triggered Communication mode enabled

Note: For more information on Time Triggered Communication mode, please refer to Section 21.4.2: Time triggered communication mode.

Bit 6 **ABOM**: *Automatic Bus-Off Management*

This bit controls the behavior of the CAN hardware on leaving the Bus-Off state.
0: The Bus-Off state is left on software request, once 128 occurrences of 11 recessive bits have been monitored and the software has first set and cleared the INRQ bit of the CAN_MCR register.
1: The Bus-Off state is left automatically by hardware once 128 occurrences of 11 recessive bits have been monitored.
For detailed information on the Bus-Off state please refer to *Section 21.4.6: Error management*.

Bit 5 **AWUM**: *Automatic Wakeup Mode*

This bit controls the behavior of the CAN hardware on message reception during Sleep mode.
0: The Sleep mode is left on software request by clearing the SLEEP bit of the CAN_MCR register.
1: The Sleep mode is left automatically by hardware on CAN message detection.
The SLEEP bit of the CAN_MCR register and the SLAK bit of the CAN_MSR register are cleared by hardware.

Bit 4 **NART**: *No Automatic Retransmission*

0: The CAN hardware will automatically retransmit the message until it has been successfully transmitted according to the CAN standard.
1: A message will be transmitted only once, independently of the transmission result (successful, error or arbitration lost).

Bit 3 **RFLM**: *Receive FIFO Locked Mode*

0: Receive FIFO not locked on overrun. Once a receive FIFO is full the next incoming message will overwrite the previous one.
1: Receive FIFO locked against overrun. Once a receive FIFO is full the next incoming message will be discarded.

Bit 2 **TXFP**: *Transmit FIFO Priority*

This bit controls the transmission order when several mailboxes are pending at the same time.
0: Priority driven by the identifier of the message
1: Priority driven by the request order (chronologically)

Bit 1 **SLEEP**: *Sleep Mode Request*

This bit is set by software to request the CAN hardware to enter the Sleep mode. Sleep mode will be entered as soon as the current CAN activity (transmission or reception of a CAN frame) has been completed.
This bit is cleared by software to exit Sleep mode.
This bit is cleared by hardware when the AWUM bit is set and a SOF bit is detected on the CAN Rx signal.
This bit is set after reset - CAN starts in Sleep mode.

Bit 0 **INRQ**: *Initialization Request*

The software clears this bit to switch the hardware into normal mode. Once 11 consecutive recessive bits have been monitored on the Rx signal the CAN hardware is synchronized and ready for transmission and reception. Hardware signals this event by clearing the INAK bit in the CAN_MSR register.

Software sets this bit to request the CAN hardware to enter initialization mode. Once software has set the INRQ bit, the CAN hardware waits until the current CAN activity (transmission or reception) is completed before entering the initialization mode. Hardware signals this event by setting the INAK bit in the CAN_MSR register.

CAN master status register (CAN_MSR)

Address offset: 0x04
Reset value: 0x0000 0C02

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				RX	SAMP	RXM	TXM	Reserved			SLAKI	WKUI	ERRI	SLAK	INAK
				r	r	r	r				rc_w1	rc_w1	rc_w1	r	r

Bits 31:12 Reserved, forced by hardware to 0.

Bit 11 **RX**: *CAN Rx Signal*

Monitors the actual value of the **CAN_RX** Pin.

Bit 10 **SAMP**: *Last Sample Point*

The value of RX on the last sample point (current received bit value).

Bit 9 **RXM**: *Receive Mode*

The CAN hardware is currently receiver.

Bit 8 **TXM**: *Transmit Mode*

The CAN hardware is currently transmitter.

Bits 7:5 Reserved, forced by hardware to 0.

Bit 4 **SLAKI**: *Sleep Acknowledge Interrupt*

When SLKIE=1, this bit is set by hardware to signal that the bxCAN has entered Sleep Mode. When set, this bit generates a status change interrupt if the SLKIE bit in the CAN_IER register is set.

This bit is cleared by software or by hardware, when SLAK is cleared.

Note: When SLKIE=0, no polling on SLAKI is possible. In this case the SLAK bit can be polled.

Bit 3 **WKUI**: *Wakeup Interrupt*

This bit is set by hardware to signal that a SOF bit has been detected while the CAN hardware was in Sleep mode. Setting this bit generates a status change interrupt if the WKUIE bit in the CAN_IER register is set.

This bit is cleared by software.

Bit 2 **ERRI**: *Error Interrupt*

This bit is set by hardware when a bit of the CAN_ESR has been set on error detection and the corresponding interrupt in the CAN_IER is enabled. Setting this bit generates a status change interrupt if the ERRIE bit in the CAN_IER register is set.
This bit is cleared by software.

Bit 1 **SLAK**: *Sleep Acknowledge*

This bit is set by hardware and indicates to the software that the CAN hardware is now in Sleep mode. This bit acknowledges the Sleep mode request from the software (set SLEEP bit in CAN_MCR register).
This bit is cleared by hardware when the CAN hardware has left Sleep mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

Note: The process of leaving Sleep mode is triggered when the SLEEP bit in the CAN_MCR register is cleared. Please refer to the AWUM bit of the CAN_MCR register description for detailed information for clearing SLEEP bit

Bit 0 **INAK**: *Initialization Acknowledge*

This bit is set by hardware and indicates to the software that the CAN hardware is now in initialization mode. This bit acknowledges the initialization request from the software (set INRQ bit in CAN_MCR register).
This bit is cleared by hardware when the CAN hardware has left the initialization mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

CAN transmit status register (CAN_TSR)

Address offset: 0x08
Reset value: 0x1C00 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]		ABRQ 2	Reserved			TERR 2	ALST2	TXOK 2	RQCP 2
r	r	r	r	r	r	r	r	rs				rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ 1	Reserved Res.			TERR 1	ALST1	TXOK 1	RQCP 1	ABRQ 0	Reserved			TERR 0	ALST0	TXOK 0	RQCP 0
rs				rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1

Bit 31 **LOW2**: *Lowest Priority Flag for Mailbox 2*

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 2 has the lowest priority.

Bit 30 **LOW1**: *Lowest Priority Flag for Mailbox 1*

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 1 has the lowest priority.

Bit 29 **LOW0**: *Lowest Priority Flag for Mailbox 0*

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 0 has the lowest priority.

Note: The LOW[2:0] bits are set to zero when only one mailbox is pending.

Bit 28 **TME2**: *Transmit Mailbox 2 Empty*

This bit is set by hardware when no transmit request is pending for mailbox 2.

Bit 27 **TME1**: *Transmit Mailbox 1 Empty*

This bit is set by hardware when no transmit request is pending for mailbox 1.

Bit 26 **TME0**: *Transmit Mailbox 0 Empty*

This bit is set by hardware when no transmit request is pending for mailbox 0.

Bits 25:24 **CODE[1:0]**: *Mailbox Code*

In case at least one transmit mailbox is free, the code value is equal to the number of the next transmit mailbox free.
In case all transmit mailboxes are pending, the code value is equal to the number of the transmit mailbox with the lowest priority.

Bit 23 **ABRQ2**: *Abort Request for Mailbox 2*

Set by software to abort the transmission request for the corresponding mailbox.
Cleared by hardware when the mailbox becomes empty.
Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 22:20 Reserved, forced by hardware to 0.

Bit 19 **TERR2**: *Transmission Error of Mailbox 2*

This bit is set when the previous TX failed due to an error.

Bit 18 **ALST2**: *Arbitration Lost for Mailbox 2*

This bit is set when the previous TX failed due to an arbitration lost.

Bit 17 **TXOK2**: *Transmission OK of Mailbox 2*

The hardware updates this bit after each transmission attempt.
0: The previous transmission failed
1: The previous transmission was successful
This bit is set by hardware when the transmission request on mailbox 2 has been completed successfully. Please refer to *Figure 197*.

Bit 16 **RQCP2**: *Request Completed Mailbox2*

Set by hardware when the last request (transmit or abort) has been performed.
Cleared by software writing a “1” or by hardware on transmission request (TXRQ2 set in CAN_TMID2R register).
Clearing this bit clears all the status bits (TXOK2, ALST2 and TERR2) for Mailbox 2.

Bit 15 **ABRQ1**: *Abort Request for Mailbox 1*

Set by software to abort the transmission request for the corresponding mailbox.
Cleared by hardware when the mailbox becomes empty.
Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 14:12 Reserved, forced by hardware to 0.

Bit 11 **TERR1**: *Transmission Error of Mailbox1*

This bit is set when the previous TX failed due to an error.

Bit 10 **ALST1**: *Arbitration Lost for Mailbox1*

This bit is set when the previous TX failed due to an arbitration lost.

Bit 9 **TXOK1**: *Transmission OK of Mailbox1*

The hardware updates this bit after each transmission attempt.
0: The previous transmission failed
1: The previous transmission was successful
This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Please refer to *Figure 197*

Bit 8 **RQCP1**: *Request Completed Mailbox1*

Set by hardware when the last request (transmit or abort) has been performed.
Cleared by software writing a “1” or by hardware on transmission request (TXRQ1 set in CAN_TI1R register).
Clearing this bit clears all the status bits (TXOK1, ALST1 and TERR1) for Mailbox 1.

Bit 7 **ABRQ0**: *Abort Request for Mailbox0*

Set by software to abort the transmission request for the corresponding mailbox.
Cleared by hardware when the mailbox becomes empty.
Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 6:4 Reserved, forced by hardware to 0.

Bit 3 **TERR0**: *Transmission Error of Mailbox0*

This bit is set when the previous TX failed due to an error.

Bit 2 **ALST0**: *Arbitration Lost for Mailbox0*

This bit is set when the previous TX failed due to an arbitration lost.

Bit 1 **TXOK0**: *Transmission OK of Mailbox0*

The hardware updates this bit after each transmission attempt.
0: The previous transmission failed
1: The previous transmission was successful
This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Please refer to *Figure 197*

Bit 0 **RQCP0**: *Request Completed Mailbox0*

Set by hardware when the last request (transmit or abort) has been performed.
Cleared by software writing a “1” or by hardware on transmission request (TXRQ0 set in CAN_TI0R register).
Clearing this bit clears all the status bits (TXOK0, ALST0 and TERR0) for Mailbox 0.

CAN receive FIFO 0 register (CAN_RF0R)

Address offset: 0x0C
Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]	
										rs	rc_w1	rc_w1		r	r

Bit 31:6 Reserved, forced by hardware to 0.

Bit 5 **RFOM0**: *Release FIFO 0 Output Mailbox*

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.
Cleared by hardware when the output mailbox has been released.

Bit 4 **FOVR0**: *FIFO 0 Overrun*

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.
This bit is cleared by software.

Bit 3 **FULL0**: *FIFO 0 Full*

Set by hardware when three messages are stored in the FIFO.
This bit is cleared by software.

Bit 2 Reserved, forced by hardware to 0.

Bits 1:0 **FMP0[1:0]**: *FIFO 0 Message Pending*

These bits indicate how many messages are pending in the receive FIFO.
FMP is increased each time the hardware stores a new message in to the FIFO. FMP is decreased each time the software releases the output mailbox by setting the RFOM0 bit.

CAN receive FIFO 1 register (CAN_RF1R)

Address offset: 0x10
Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]	
										rs	rc_w1	rc_w1		r	r

Bits 31:6 Reserved, forced by hardware to 0.

Bit 5 **RFOM1**: *Release FIFO 1 Output Mailbox*

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.
Cleared by hardware when the output mailbox has been released.

Bit 4 **FOVR1**: *FIFO 1 Overrun*

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.
This bit is cleared by software.

Bit 3 **FULL1**: *FIFO 1 Full*

Set by hardware when three messages are stored in the FIFO.
This bit is cleared by software.

Bit 2 Reserved, forced by hardware to 0.

Bits 1:0 **FMP1[1:0]**: *FIFO 1 Message Pending*

These bits indicate how many messages are pending in the receive FIFO1.
FMP1 is increased each time the hardware stores a new message in to the FIFO1. FMP is decreased each time the software releases the output mailbox by setting the RFOM1 bit.

CAN interrupt enable register (CAN_IER)

Address offset: 0x14
Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														SLKIE	WKUIE
														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Reserved			LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE
rw				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:18 Reserved, forced by hardware to 0.

Bit 17 **SLKIE**: Sleep *Interrupt Enable*
0: No interrupt when SLAKI bit is set.
1: Interrupt generated when SLAKI bit is set.

Bit 16 **WKUIE**: *Wakeup Interrupt Enable*
0: No interrupt when WKUI is set.
1: Interrupt generated when WKUI bit is set.

Bit 15 **ERRIE**: *Error Interrupt Enable*
0: No interrupt will be generated when an error condition is pending in the CAN_ESR.
1: An interrupt will be generation when an error condition is pending in the CAN_ESR.

Bits 14:12 Reserved, forced by hardware to 0.

Bit 11 **LECIE**: *Last Error Code Interrupt Enable*
0: ERRI bit will not be set when the error code in LEC[2:0] is set by hardware on error detection.
1: ERRI bit will be set when the error code in LEC[2:0] is set by hardware on error detection.

Bit 10 **BOFIE**: *Bus-Off Interrupt Enable*
0: ERRI bit will not be set when BOFF is set.
1: ERRI bit will be set when BOFF is set.

Bit 9 **EPVIE**: *Error Passive Interrupt Enable*
0: ERRI bit will not be set when EPVF is set.
1: ERRI bit will be set when EPVF is set.

Bit 8 **EWGIE**: *Error Warning Interrupt Enable*
0: ERRI bit will not be set when EWGF is set.
1: ERRI bit will be set when EWGF is set.

Bit 7 Reserved, forced by hardware to 0.

Bit 6 **FOVIE1**: *FIFO Overrun Interrupt Enable*
0: No interrupt when FOVR is set.
1: Interrupt generation when FOVR is set.

Bit 5 **FFIE1**: *FIFO Full Interrupt Enable*
0: No interrupt when FULL bit is set.
1: Interrupt generated when FULL bit is set.

Bit 4 **FMPIE1**: *FIFO Message Pending Interrupt Enable*
0: No interrupt generated when state of FMP[1:0] bits are not 00b.
1: Interrupt generated when state of FMP[1:0] bits are not 00b.

Bit 3 **FOVIE0**: *FIFO Overrun Interrupt Enable*

0: No interrupt when FOVR bit is set.
1: Interrupt generated when FOVR bit is set.

Bit 2 **FFIE0**: *FIFO Full Interrupt Enable*

0: No interrupt when FULL bit is set.
1: Interrupt generated when FULL bit is set.

Bit 1 **FMPIE0**: *FIFO Message Pending Interrupt Enable*

0: No interrupt generated when state of FMP[1:0] bits are not 00b.
1: Interrupt generated when state of FMP[1:0] bits are not 00b.

Bit 0 **TMEIE**: *Transmit Mailbox Empty Interrupt Enable*

0: No interrupt when RQCPx bit is set.
1: Interrupt generated when RQCPx bit is set.

Note: refer to *Section 21.5: bxCAN interrupts*.

CAN error status register (CAN_ESR)

Address offset: 0x18
Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REC[7:0]								TEC[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									LEC[2:0]			Res.	BOFF	EPVF	EWGF
									rw	rw	rw		r	r	r

Bits 31:24 **REC[7:0]**: *Receive Error Counter*

The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

Bits 23:16 **TEC[7:0]**: *least significant byte of the 9-bit Transmit Error Counter*

The implementing part of the fault confinement mechanism of the CAN protocol.

Bits 15:7 Reserved, forced by hardware to 0.

Bits 6:4 **LEC[2:0]**: *Last Error Code*

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to ‘0’.
Code 7 is unused and may be written by the hardware to check for an update

000: No Error
001: Stuff Error
010: Form Error
011: Acknowledgment Error
100: Bit recessive Error
101: Bit dominant Error
110: CRC Error
111: Set by software

Bit 3 Reserved, forced by hardware to 0.

Bit 2 **BOFF**: *Bus-Off Flag*

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255, refer to *Section 21.4.6 on page 504*.

Bit 1 **EPVF**: *Error Passive Flag*

This bit is set by hardware when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter>127).

Bit 0 **EWGF**: *Error Warning Flag*

This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter≥96).

CAN bit timing register (CAN_BTR)

Address offset: 0x1C
Reset value: 0x0123 0000

Note: This register can only be accessed by the software when the CAN hardware is in initialization mode.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SILM	LBKM	Reserved				SJW[1:0]		Res.	TS2[2:0]			TS1[3:0]			
rw	rw					rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						BRP[9:0]									
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **SILM**: *Silent Mode (Debug)*

0: Normal operation
1: Silent Mode

Bit 30 **LBKM**: *Loop Back Mode (Debug)*

0: Loop Back Mode disabled
1: Loop Back Mode enabled

Bits 29:26 Reserved, forced by hardware to 0.

Bits 25:24 **SJW[1:0]**: *Resynchronization Jump Width*

These bits define the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform the resynchronization.

$t_{RJW} = t_{CAN} \times (SJW[1:0] + 1)$

Bit 23 Reserved, forced by hardware to 0.

Bits 22:20 **TS2[2:0]**: *Time Segment 2*

These bits define the number of time quanta in Time Segment 2.

$t_{BS2} = t_{CAN} \times (TS2[2:0] + 1)$

Bits 19:16 **TS1[3:0]**: *Time Segment 1*

These bits define the number of time quanta in Time Segment 1

$t_{BS1} = t_{CAN} \times (TS1[3:0] + 1)$

For more information on bit timing, please refer to *Section 21.4.7: Bit timing on page 504*.

Bits 15:10 Reserved, forced by hardware to 0.

Bits 9:0 **BRP[9:0]**: *Baud Rate Prescaler*

These bits define the length of a time quanta.

$t_q = (BRP[9:0]+1) \times t_{PCLK}$

21.6.3 Mailbox registers

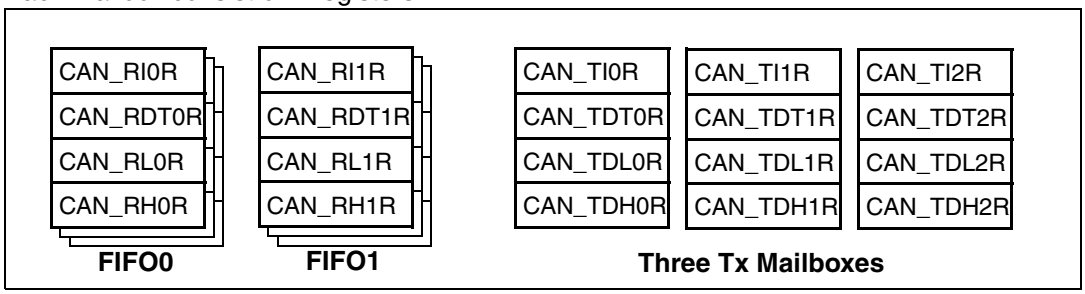
This chapter describes the registers of the transmit and receive mailboxes. Refer to *Section 21.4.5: Message storage on page 502* for detailed register mapping.

Transmit and receive mailboxes have the same registers except:

- The FMI field in the CAN_RDTxR register.
- A receive mailbox is always write protected.
- A transmit mailbox is write-enabled only while empty, corresponding TME bit in the CAN_TSR register set.

There are 3 TX Mailboxes and 2 RX Mailboxes. Each RX Mailbox allows access to a 3 level depth FIFO, the access being offered only to the oldest received message in the FIFO.

Each mailbox consist of 4 registers.



TX mailbox identifier register (CAN_TIxR) (x=0..2)

Address offsets: 0x180, 0x190, 0x1A0
Reset value: undefined (except bit 0, TXRQ = 0)

Note: 1 All TX registers are write protected when the mailbox is pending transmission (TMEx reset).

2 This register also implements the TX request control (bit 0) - reset value 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	TXRQ
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:21 **STID[10:0]/EXID[28:18]**: *Standard Identifier or Extended Identifier*

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bit 20:3 **EXID[17:0]**: *Extended Identifier*

The LSBs of the extended identifier.

Bit 2 **IDE**: *Identifier Extension*

This bit defines the identifier type of message in the mailbox.
0: Standard identifier.
1: Extended identifier.

Bit 1 **RTR**: *Remote Transmission Request*

0: Data frame
1: Remote frame

Bit 0 **TXRQ**: *Transmit Mailbox Request*

Set by software to request the transmission for the corresponding mailbox.
Cleared by hardware when the mailbox becomes empty.

Mailbox data length control and time stamp register (CAN_TDTxR) (x=0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x184, 0x194, 0x1A4
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							TGT	Reserved				DLC[3:0]			
							rw					rw	rw	rw	rw

Bits 31:16 **TIME[15:0]**: *Message Time Stamp*

This field contains the 16-bit timer value captured at the SOF transmission.

Bits 15:9 Reserved

Bit 8 **TGT**: *Transmit Global Time*

This bit is active only when the hardware is in the Time Trigger Communication mode, TTCM bit of the CAN_MCR register is set.

0: Time stamp TIME[15:0] is not sent.

1: Time stamp TIME[15:0] value is sent in the last two data bytes of the 8-byte message: TIME[7:0] in data byte 6 and TIME[15:8] in data byte 7, replacing the data written in CAN_TDHxR[31:16] register (DATA6[7:0] and DATA7[7:0]). DLC must be programmed as 8 in order these two bytes to be sent over the CAN bus.

Bits 7:4 Reserved

Bits 3:0 **DLC[3:0]**: *Data Length Code*

This field defines the number of data bytes a data frame contains or a remote frame request. A message can contain from 0 to 8 data bytes, depending on the value in the DLC field.

Mailbox data low register (CAN_TDLxR) (x=0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x188, 0x198, 0x1A8
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA3[7:0]**: *Data Byte 3*
Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: *Data Byte 2*
Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: *Data Byte 1*
Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]**: *Data Byte 0*
Data byte 0 of the message.
A message can contain from 0 to 8 data bytes and starts with byte 0.

Mailbox data high register (CAN_TDHxR) (x=0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x18C, 0x19C, 0x1AC
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA7[7:0]**: *Data Byte 7*
Data byte 7 of the message.
Note: if TGT of this message and TTCM are active, DATA7 and DATA6 will be replaced by the TIME stamp value.

Bits 23:16 **DATA6[7:0]**: *Data Byte 6*
Data byte 6 of the message.

Bits 15:8 **DATA5[7:0]**: *Data Byte 5*
Data byte 5 of the message.

Bits 7:0 **DATA4[7:0]**: *Data Byte 4*
Data byte 4 of the message.

Rx FIFO mailbox identifier register (CAN_RIxR) (x=0..1)

Address offsets: 0x1B0, 0x1C0
Reset value: undefined

Note: All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	Res.
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bits 31:21 **STID[10:0]/EXID[28:18]**: *Standard Identifier or Extended Identifier*
The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bits 20:3 **EXID[17:0]**: *Extended Identifier*
The LSBs of the extended identifier.

Bit 2 **IDE**: Identifier Extension
This bit defines the identifier type of message in the mailbox.
0: Standard identifier.
1: Extended identifier.

Bit 1 **RTR**: *Remote Transmission Request*
0: Data frame
1: Remote frame

Bit 0 Reserved

Receive FIFO mailbox data length control and time stamp register (CAN_RDTxR) (x=0..1)

Address offsets: 0x1B4, 0x1C4
Reset value: undefined

Note: All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FMI[7:0]								Reserved				DLC[3:0]			
r	r	r	r	r	r	r	r					r	r	r	r

Bits 31:16 **TIME[15:0]**: *Message Time Stamp*
This field contains the 16-bit timer value captured at the SOF detection.

Bits 15:8 **FMI[7:0]**: *Filter Match Index*
This register contains the index of the filter the message stored in the mailbox passed through. For more details on identifier filtering please refer to *Section 21.4.4: Identifier filtering on page 498* - **Filter Match Index** paragraph.

Bits 7:4 Reserved, forced by hardware to 0.

Bits 3:0 **DLC[3:0]**: *Data Length Code*
This field defines the number of data bytes a data frame contains (0 to 8). It is 0 in the case of a remote frame request.

Receive FIFO mailbox data low register (CAN_RDLxR) (x=0..1)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x1B8, 0x1C8
Reset value: undefined

Note: All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA3[7:0]**: *Data Byte 3*
Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: *Data Byte 2*
Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: *Data Byte 1*
Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]**: *Data Byte 0*
Data byte 0 of the message.
A message can contain from 0 to 8 data bytes and starts with byte 0.

Receive FIFO mailbox data high register (CAN_RDHxR) (x=0..1)

Address offsets: 0x1BC, 0x1CC
Reset value: undefined

Note: All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA7[7:0]**: *Data Byte 7*
Data byte 3 of the message.

Bits 23:16 **DATA6[7:0]**: *Data Byte 6*
Data byte 2 of the message.

Bits 15:8 **DATA5[7:0]**: *Data Byte 5*
Data byte 1 of the message.

Bits 7:0 **DATA4[7:0]**: *Data Byte 4*
Data byte 0 of the message.

21.6.4 CAN filter registers

CAN filter master register (CAN_FMR)

Address offset: 0x200
Reset value: 0x2A1C 0E01

Note: All bits of this register are set and cleared by software.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															FINIT
															rw
Reserved		CAN2SB[5:0]						Reserved							FINIT
		rw	rw	rw	rw	rw	rw								rw

Bits 31:1 Reserved, forced to reset value

Bit 0 **FINIT**: *Filter Init Mode*
Initialization mode for filter banks
0: Active filters mode.
1: Initialization mode for the filters.

CAN filter mode register (CAN_FM1R)

Address offset: 0x204
Reset value: 0x00

Note: This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		FBM13	FBM12	FBM11	FBM10	FBM9	FBM8	FBM7	FBM6	FBM5	FBM4	FBM3	FBM2	FBM1	FBM0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: Please refer to Figure 199: Filter bank scale configuration - register organization on page 500

Bits 31:14 Reserved. Forced to 0 by hardware.

Bits 13:0 **FBM*x***: *Filter Mode*

Note: Mode of the registers of Filter x.
0: Two 32-bit registers of filter bank x are in Identifier Mask mode.
1: Two 32-bit registers of filter bank x are in Identifier List mode.

CAN filter scale register (CAN_FS1R)

Address offset: 0x20C
Reset value: 0x00

Note: This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		FSC13	FSC12	FSC11	FSC10	FSC9	FSC8	FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: Please refer to Figure 199: Filter bank scale configuration - register organization on page 500

Bits 31:14 Reserved, forced by hardware to 0.

Bits 13:0 **FSCx**: *Filter Scale Configuration*

These bits define the scale configuration of Filters 13-0.
0: Dual 16-bit scale configuration
Note: 1: Single 32-bit scale configuration

CAN filter FIFO assignment register (CAN_FFA1R)

Address offset: 0x214
Reset value: 0x00

Note: This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		FFA13	FFA12	FFA11	FFA10	FFA9	FFA8	FFA7	FFA6	FFA5	FFA4	FFA3	FFA2	FFA1	FFA0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:14 Reserved, forced by hardware to 0.

Bits 13:0 **FFA*x***: *Filter FIFO Assignment for Filter x*

Note: The message passing through this filter will be stored in the specified FIFO.
0: Filter assigned to FIFO 0
1: Filter assigned to FIFO 1

CAN filter activation register (CAN_FA1R)

Address offset: 0x21C
Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		FACT13	FACT12	FACT11	FACT10	FACT9	FACT8	FACT7	FACT6	FACT5	FACT4	FACT3	FACT2	FACT1	FACT0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:14 Reserved, forced by hardware to 0.

Bits 13:0 **FACT*x***: *Filter Active*

Note: The software sets this bit to activate Filter x. To modify the Filter x registers (CAN_FxR[0:7]), the FACTx bit must be cleared or the FINIT bit of the CAN_FMR register must be set.
0: Filter x is not active
1: Filter x is active

Filter bank i register x (CAN_FiRx) (i=0..13, x=1..2)

Address offsets: 0x240..0x2AC
Reset value: undefined

Note: There are 14 filter banks, i=0..13. Each filter bank i is composed of two 32-bit registers, CAN_FiR[2:1].

This register can only be modified when the FACTx bit of the CAN_FAxR register is cleared or when the FINIT bit of the CAN_FMR register is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FB31	FB30	FB29	FB28	FB27	FB26	FB25	FB24	FB23	FB22	FB21	FB20	FB19	FB18	FB17	FB16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FB15	FB14	FB13	FB12	FB11	FB10	FB9	FB8	FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

In all configurations:

Bits 31:0 **FB[31:0]**: *Filter Bits*

Identifier

Each bit of the register specifies the level of the corresponding bit of the expected identifier.
0: Dominant bit is expected
1: Recessive bit is expected

Mask

Each bit of the register specifies whether the bit of the associated identifier register must match with the corresponding bit of the expected identifier or not.

0: Don't care, the bit is not used for the comparison

1: Must match, the bit of the incoming identifier must have the same level has specified in the corresponding identifier register of the filter.

Note: Depending on the scale and mode configuration of the filter the function of each register can differ. For the filter mapping, functions description and mask registers association, refer to Section 21.4.4: Identifier filtering on page 498.

A Mask/Identifier register in **mask mode** has the same bit mapping as in **identifier list** mode.

For the register mapping/addresses of the filter banks please refer to the Table 146 on page 528.

21.6.5 bxCAN register map

Refer to *Table 1 on page 35* for the register boundary addresses.

Table 146. bxCAN register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000	CAN_MCR	Reserved																RESET	Reserved							TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
	Reset value																	0								0	0	0	0	0	0	1	0
0x004	CAN_MSR	Reserved																				RX	SAMP	RXM	TXM	Reserved			SLAKI	WKUI	ERRI	SLAK	INAK
	Reset value																					1	1	0	0				0	0	0	1	0
0x008	CAN_TSR	LOW[2:0]			TME[2:0]			CODE[1:0]		ABRQ2	Reserved			TERR2	ALST2	TXOK2	RQCP2	ABRQ1	Reserved			TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Reserved			TERR0	ALST0	TXOK0	RQCP0
	Reset value	0	0	0	1	1	1	0	0	0				0	0	0	0	0				0	0	0	0	0				0	0	0	0
0x00C	CAN_RF0R	Reserved																										RFOM0	FOVR0	FULL0	Reserved	FMP0[1:0]	
	Reset value																											0	0	0		0	0
0x010	CAN_RF1R	Reserved																										RFOM1	FOVR1	FULL1	Reserved	FMP1[1:0]	
	Reset value																											0	0	0		0	0
0x014	CAN_IER	Reserved														SLKIE	WKUIE	ERRIE	Reserved			LECIE	BOFIE	EPVIE	EWGIE	Reserved	FOVIE1	FFIE1	FMPIE1	FOVIE0	FFIE0	FMPIE0	TMEIE
	Reset value															0	0	0				0	0	0	0		0	0	0	0	0	0	0
0x018	CAN_ESR	REC[7:0]								TEC[7:0]								Reserved									LEC[2:0]			Reserved	BOFF	EPVF	EWGF
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										0	0	0		0	0	0
0x01C	CAN_BTR	SILM	LBKM	Reserved				SJW[1:0]		Reserved	TS2[2:0]			TS1[3:0]				Reserved						BRP[9:0]									
	Reset value	0	0					0	0		0	1	0	0	0	1	1							0	0	0	0	0	0	0	0	0	0
0x020-0x17F	Reserved																																
0x180	CAN_TI0R	STID[10:0]/EXID[28:18]											EXID[17:0]																		IDE	RTR	TXRQ
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0
0x184	CAN_TDT0R	TIME[15:0]																Reserved							TGT	Reserved				DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								x					x	x	x	x
0x188	CAN_TDL0R	DATA3[7:0]								DATA2[7:0]								DATA1[7:0]								DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x18C	CAN_TDH0R	DATA7[7:0]								DATA6[7:0]								DATA5[7:0]								DATA4[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x190	CAN_TI1R	STID[10:0]/EXID[28:18]											EXID[17:0]																		IDE	RTR	TXRQ
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0
0x194	CAN_TDT1R	TIME[15:0]																Reserved							TGT	Reserved				DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								x					x	x	x	x

Table 146. bxCAN register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x198	**CAN_TDL1R**	DATA3[7:0]								DATA2[7:0]								DATA1[7:0]								DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x19C	**CAN_TDH1R**	DATA7[7:0]								DATA6[7:0]								DATA5[7:0]								DATA4[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1A0	**CAN_TI2R**	STID[10:0]/EXID[28:18]											EXID[17:0]																		IDE	RTR	TXRQ
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0
0x1A4	**CAN_TDT2R**	TIME[15:0]																Reserved							TGT	Reserved					DLC[3:0]		
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								x					x	x	x	x
0x1A8	**CAN_TDL2R**	DATA3[7:0]								DATA2[7:0]								DATA1[7:0]								DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1AC	**CAN_TDH2R**	DATA7[7:0]								DATA6[7:0]								DATA5[7:0]								DATA4[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1B0	**CAN_RI0R**	STID[10:0]/EXID[28:18]											EXID[17:0]																		IDE	RTR	Reserved
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
0x1B4	**CAN_RDT0R**	TIME[15:0]																FMI[7:0]								Reserved					DLC[3:0]		
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x
0x1B8	**CAN_RDL0R**	DATA3[7:0]								DATA2[7:0]								DATA1[7:0]								DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1BC	**CAN_RDH0R**	DATA7[7:0]								DATA6[7:0]								DATA5[7:0]								DATA4[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1C0	**CAN_RI1R**	STID[10:0]/EXID[28:18]											EXID[17:0]																		IDE	RTR	Reserved
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
0x1C4	**CAN_RDT1R**	TIME[15:0]																FMI[7:0]								Reserved					DLC[3:0]		
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x
0x1C8	**CAN_RDL1R**	DATA3[7:0]								DATA2[7:0]								DATA1[7:0]								DATA0[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1CC	**CAN_RDH1R**	DATA7[7:0]								DATA6[7:0]								DATA5[7:0]								DATA4[7:0]							
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x1D0-0x1FF	Reserved																																
0x200	**CAN_FMR**	Reserved																															FINIT
	Reset value																																1
0x204	**CAN_FM1R**	Reserved																		FBM[13:0]													
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 146. bxCAN register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x208	Reserved																																
0x20C	**CAN_FS1R**	Reserved																		FSC[13:0]													
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x210	Reserved																																
0x214	**CAN_FFA1R**	Reserved																		FFA[13:0]													
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x218	Reserved																																
0x21C	**CAN_FA1R**	Reserved																		FACT[13:0]													
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x220	Reserved																																
0x224-0x23F	Reserved																																
0x240	**CAN_F0R1**	FB[31:0]																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x244	**CAN_F0R2**	FB[31:0]																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x248	**CAN_F1R1**	FB[31:0]																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x24C	**CAN_F1R2**	FB[31:0]																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
. . . .	. . . .	. . . .																															
0x2A8	**CAN_F13R1**	FB[31:0]																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x2AC	**CAN_F13R2**	FB[31:0]																															
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

22 Serial peripheral interface (SPI)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

22.1 SPI introduction

In high-density devices, the SPI interface gives the flexibility to get either the SPI protocol or the I^2S audio protocol. By default, it is the SPI function that is selected. It is possible to switch the interface from SPI to I^2S by software.
In low- and medium-density devices, the I^2S protocol is not available.

The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multimaster configuration.

It may be used for a variety of purposes, including Simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication using CRC checking.

I^2S is also a synchronous, serial communication interface with a 3-pin protocol. It can address four different audio standards including the I^2S Phillips standard, the MSB- and LSB-justified standards and the PCM standard. It can operate in slave or master mode with half-duplex communication. Master clock may be provided by the interface to an external slave component when the I^2S is configured as the communication master.

Warning: Since some SPI3/I2S3 pins are shared with JTAG pins (SPI3_NSS/I2S3_WS with JTDI and SPI3_SCK/I2S3_CK with JTDO), they are not controlled by the I/O controller and are reserved for JTAG usage (after each Reset).
For this purpose prior to configure the SPI3/I2S3 pins, the user has to disable the JTAG and use the SWD interface (when debugging the application), or disable both JTAG/SWD interfaces (for standalone application). For more information on the configuration of JTAG/SWD interface pins, please refer to *Section 7.3.4: JTAG/SWD alternate function remapping*.

22.2 SPI and I^{2}S main features

22.2.1 SPI features

- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- Multimaster mode capability
- 8 master mode baud rate prescalers (f_{PCLK}/2 max.)
- Slave mode frequency (f_{PCLK}/2 max)
- Faster communication for both master and slave: maximum SPI speed up to 18 MHz
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Hardware CRC feature for reliable communication:
  - CRC value can be transmitted as last byte in Tx mode
  - Automatic CRC error checking for last received byte
- Master mode fault, overrun and CRC error flags with interrupt capability
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

22.2.2 I^2S features

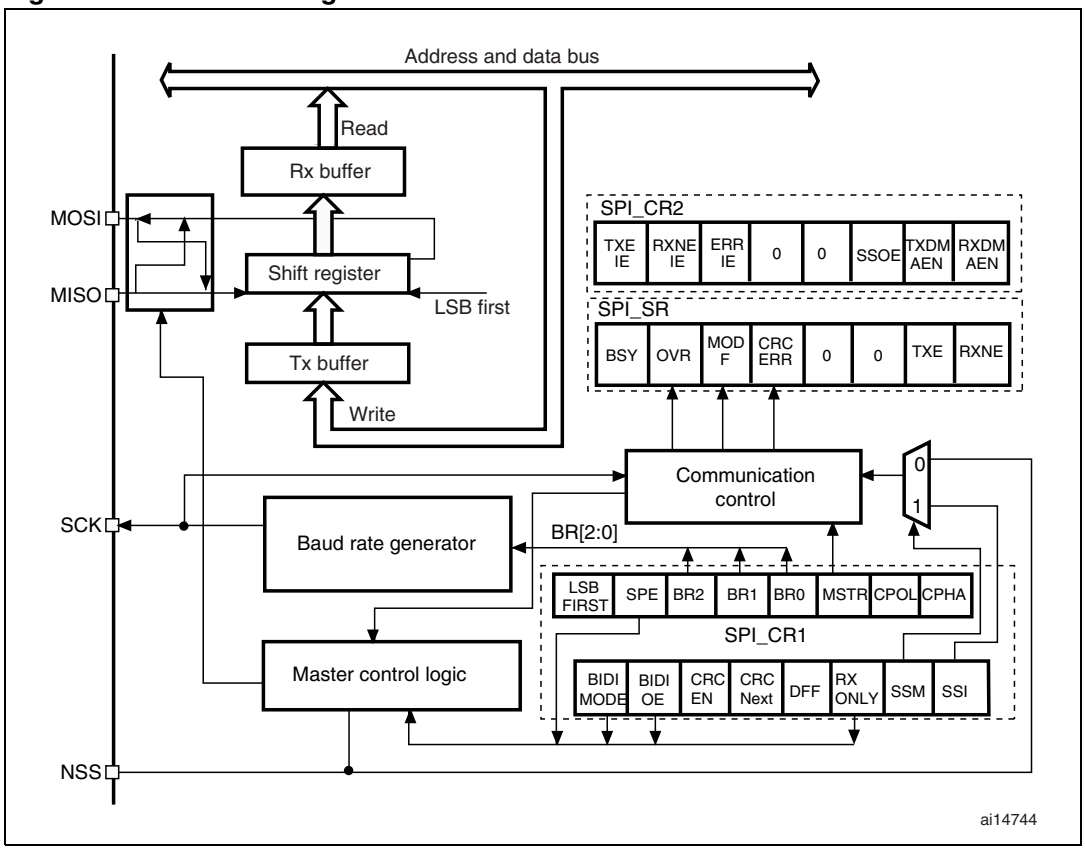
- Simplex communication (only transmitter or receiver)
- Master or slave operations
- 8-bit programmable linear prescaler to reach accurate audio sample frequencies (from 8 kHz to 48 kHz)
- Data format may be 16-bit, 24-bit or 32-bit
- Packet frame is fixed to 16-bit (16-bit data frame) or 32-bit (16-bit, 24-bit, 32-bit data frame) by audio channel
- Programmable clock polarity (steady state)
- Underrun flag in slave transmission mode and Overrun flag in reception mode (master and slave)
- 16-bit register for transmission and reception with one data register for both channel sides
- Supported I^2S protocols:
  - I^2S Phillips standard
  - MSB-Justified standard (Left-Justified)
  - LSB-Justified standard (Right-Justified)
  - PCM standard (with short and long frame synchronization on 16-bit channel frame or 16-bit data frame extended to 32-bit channel frame)
- Data direction is always MSB first
- DMA capability for transmission and reception (16-bit wide)
- Master clock may be output to drive an external audio component. Ratio is fixed at $256 \times F_S$ (where F_S is the audio sampling frequency)

22.3 SPI functional description

22.3.1 General description

The block diagram of the SPI is shown in *Figure 206*.

Figure 206. SPI block diagram

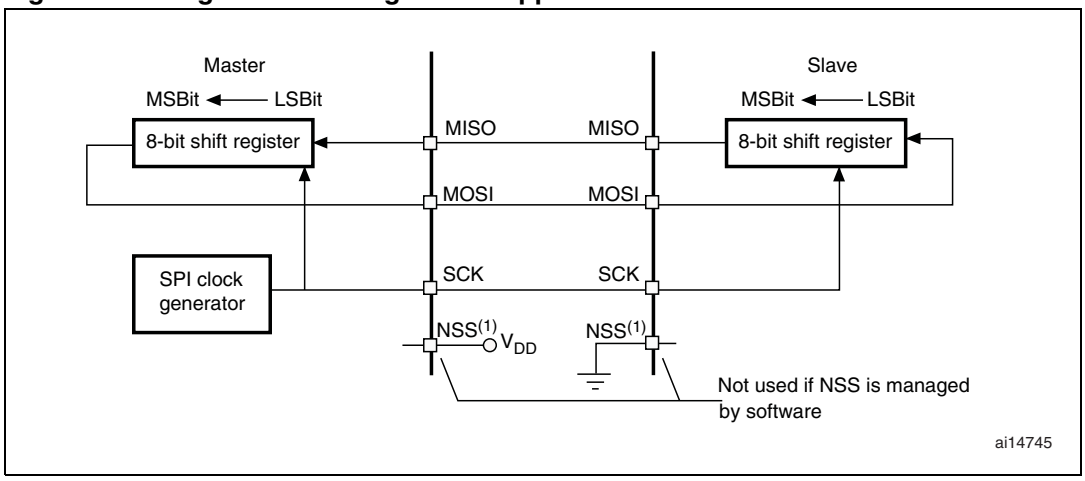


Usually, the SPI is connected to external devices through 4 pins:

- MISO: Master In / Slave Out data. This pin can be used to transmit data in slave mode and receive data in master mode.
- MOSI: Master Out / Slave In data. This pin can be used to transmit data in master mode and receive data in slave mode.
- SCK: Serial Clock output for SPI masters and input for SPI slaves.
- NSS: Slave select. This is an optional pin to select master/ slave mode. This pin acts as a ‘chip select’ to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave NSS inputs can be driven by standard I/O ports on the master Device. The NSS pin may also be used as an output if enabled (SSOE bit) and driven low if the SPI is in master configuration. In this manner, all NSS pins from devices connected to the Master NSS pin see a low level and become slaves when they are configured in NSS hardware mode.

A basic example of interconnections between a single master and a single slave is illustrated in *Figure 207*.

Figure 207. Single master/ single slave application



1. Here, the NSS pin is configured as an input.

The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

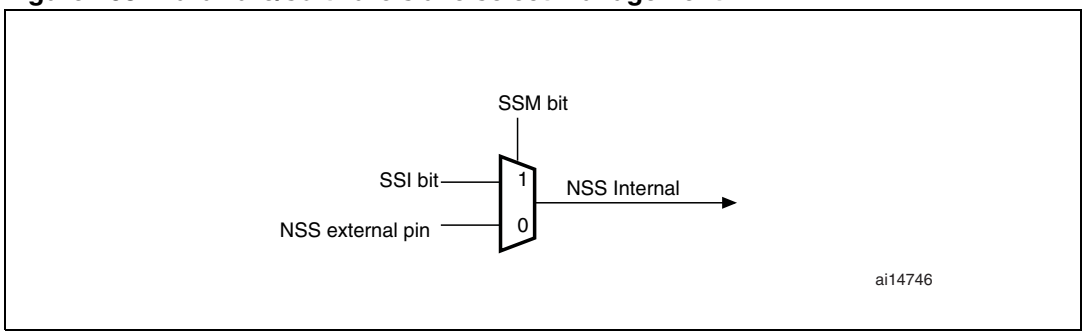
The communication is always initiated by the master. When the master device transmits data to a slave device via the MOSI pin, the slave device responds via the MISO pin. This implies full-duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

Slave select (NSS) pin management

There are two NSS modes:

- Software NSS mode: this mode is enabled by setting the SSM bit in the SPI_CR1 register (see *Figure 208*). In this mode, the external NSS pin is free for other application uses and the internal NSS signal level is driven by writing to the SSI bit in the SPI_CR1 register.
- Hardware NSS mode: there are two cases:
  - NSS output is enabled: when the STM32F10xxx is operating as a Master and the NSS output is enabled through the SSOE bit in the SPI_CR2 register, the NSS pin is driven low and all the NSS pins of devices connected to the Master NSS pin see a low level and become slaves when they are configured in NSS hardware mode. In this case, the cell cannot work in a multimaster environment.
  - NSS output is disabled: the multimaster capability is allowed.

Figure 208. Hardware/software slave select management



Clock phase and clock polarity

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPI_CR1 register. The CPOL (clock polarity) bit controls the steady state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

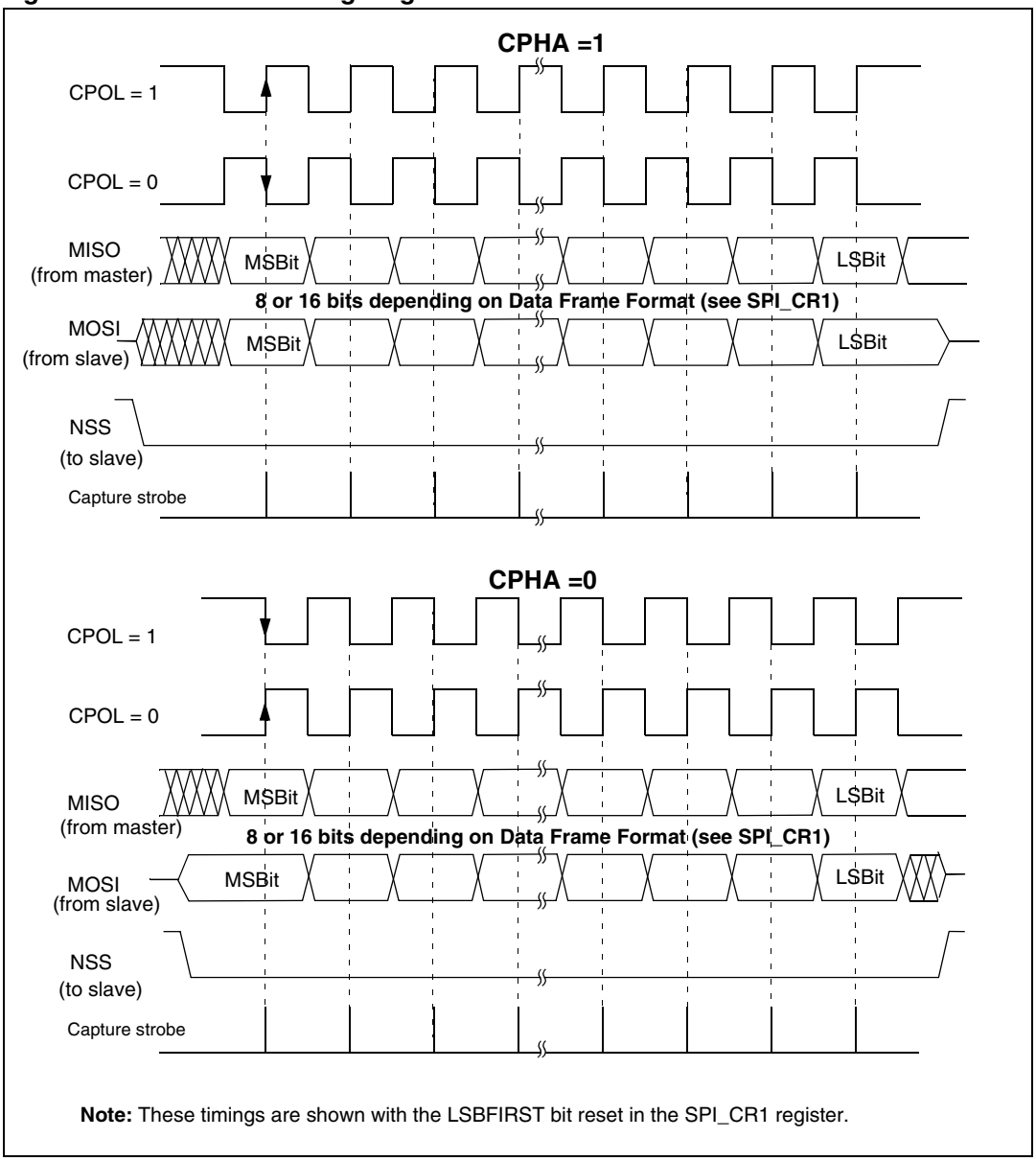
If the CPHA (clock phase) bit is set, the second edge on the SCK pin (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set) is the MSBit capture strobe. Data are latched on the occurrence of the second clock transition. If the CPHA bit is reset, the first edge on the SCK pin (falling edge if CPOL bit is set, rising edge if CPOL bit is reset) is the MSBit capture strobe. Data are latched on the occurrence of the first clock transition.

The combination of the CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

Figure 209, shows an SPI transfer with the four combinations of the CPHA and CPOL bits. The diagram may be interpreted as a master or slave timing diagram where the SCK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.

Note:
1. *Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.*
2. *Master and slave must be programmed with the same timing mode.*
3. *The idle state of SCK must correspond to the polarity selected in the SPI_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).*
4. *The Data Frame Format (8- or 16-bit) is selected through the DFF bit in SPI_CR1 register, and determines the data length during transmission/reception.*

Figure 209. Data clock timing diagram



1. These timings are shown with the LSBFIRST bit reset in the SPI_CR1 register.

Data frame format

Data can be shifted out either MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 Register.

Each data frame is 8 or 16 bits long depending on the size of the data programmed using the DFF bit in the SPI_CR1 register. The selected data frame format is applicable for transmission and/or reception.

22.3.2 SPI slave mode

In slave configuration, the serial clock is received on the SCK pin from the master device. The value set in the BR[2:0] bits in the SPI_CR1 register, does not affect the data transfer rate.

Procedure

1. Set the DFF bit to define 8- or 16-bit data frame format
2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see *Figure 209*). For correct data transfer, the CPOL and CPHA bits must be configured in the same way in the slave device and the master device.
3. The frame format (MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 register) must be the same as the master device.
4. In Hardware mode (refer to *Slave select (NSS) pin management on page 535*), the NSS pin must be connected to a low level signal during the complete byte transmit sequence. In Software mode, set the SSM bit and clear the SSI bit in the SPI_CR1 register.
5. Clear the MSTR bit and set the SPE bit (both in the SPI_CR1 register) to assign the pins to alternate functions.

In this configuration the MOSI pin is a data input and the MISO pin is a data output.

Transmit sequence

The data byte is parallel-loaded into the Tx buffer during a write cycle.

The transmit sequence begins when the slave device receives the clock signal and the most significant bit of the data on its MOSI pin. The remaining bits (the 7 bits in 8-bit data frame format, and the 15 bits in 16-bit data frame format) are loaded into the shift-register. The TXE flag in the SPI_SR register is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

Receive sequence

For the receiver, when data transfer is complete:

- The Data in shift register is transferred to Rx Buffer and the RXNE flag (SPI_SR register) is set
- An Interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register.

After the last sampling clock edge the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value.

Clearing of the RXNE bit is performed by reading the SPI_DR register.

22.3.3 SPI master mode

In the master configuration, the serial clock is generated on the SCK pin.

Procedure

1. Select the BR[2:0] bits to define the serial clock baud rate (see SPI_CR1 register).
2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see *Figure 209*).
3. Set the DFF bit to define 8- or 16-bit data frame format
4. Configure the LSBFIRST bit in the SPI_CR1 register to define the frame format
5. If the NSS pin is required in input mode, in hardware mode, connect the NSS pin to a high-level signal during the complete byte transmit sequence. In software mode, set the SSM and SSI bits in the SPI_CR1 register.
   If the NSS pin is required in output mode, the SSOE bit only should be set.
6. The MSTR and SPE bits must be set (they remain set only if the NSS pin is connected to a high-level signal).

In this configuration the MOSI pin is a data output and the MISO pin is a data input.

Transmit sequence

The transmit sequence begins when a byte is written in the Tx Buffer.

The data byte is parallel-loaded into the shift register (from the internal bus) during the first bit transmission and then shifted out serially to the MOSI pin MSB first or LSB first depending on the LSBFIRST bit in the SPI_CR1 register. The TXE flag is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

Receive sequence

For the receiver, when data transfer is complete:

- The data in the shift register is transferred to the RX Buffer and the RXNE flag is set
- An interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register

At the last sampling clock edge the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value.

Clearing the RXNE bit is performed by reading the SPI_DR register.

A continuous transmit stream can be maintained if the next data to be transmitted is put in the Tx buffer once the transmission is started. Note that TXE flag should be ‘1’ before any attempt to write the Tx buffer is made.

22.3.4 Simplex communication

The SPI is capable of operating in simplex mode in 2 configurations.

- 1 clock and 1 bidirectional data wire
- 1 clock and 1 data wire (receive-only in full-duplex mode)

1 clock and 1 bidirectional data wire

This mode is enabled by setting the BIDIMODE bit in the SPI_CR1 register. In this mode SCK is used for the clock and MOSI in master or MISO in slave mode is used for data communication. The transfer direction (Input/Output) is selected by the BIDIOE bit in the SPI_CR2 register. When this bit is 1, the data line is output otherwise it is input.

1 clock and 1 data wire (receive-only in full-duplex mode)

In order to free an I/O pin so it can be used for other purposes, it is possible to disable the SPI output function by setting the RXONLY bit in the SPI_CR1 register. In this case, SPI will function in Receive-only mode. When the RXONLY bit is reset, the SPI will function in full duplex mode.

To start the communication in receive-only mode, it is necessary to enable the SPI. In the master mode, the communication starts immediately and will stop when the SPE bit is reset and the current reception terminates. In slave mode, the SPI will continue to receive as long as the NSS is pulled down (or the SSI bit is reset) and the SCK is running.

Note: The SPI can be used in Tx-only mode when the RXONLY bit in the SPI_CR1 register is reset, the RX pin (MISO in master or MOSI in slave) can be used as GPIO. In this case, when the data register is read, it does not contain the received value.

22.3.5 Status flags

Three status flags are provided for the application to completely monitor the state of the SPI bus.

BUSY flag

This flag indicates the state of the SPI communication layer. When it is set, it indicates that the SPI is busy communicating and/or there is a valid data byte in the Tx buffer waiting to be transmitted. The purpose of this flag is to indicate if there is any communication ongoing on the SPI bus or not. This flag is set as soon as:

1. Data is written in the SPI_DR register in master mode
2. The SCK clock is present in slave mode

The BUSY flag is reset each time a byte is transmitted/received. This flag is set and cleared by hardware. It can be monitored to avoid write collision errors. Writing to this flag has no effect. The BUSY flag is meaningful only when the SPE bit is set.

Note: In master receiver mode (1-line bidirectional), the BUSY flag must NOT be checked.

Tx buffer empty flag (TXE)

When it is set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can be loaded into the buffer. The TXE flag is reset when the Tx buffer already contains data to be transmitted. This flag is reset when the SPI is disabled (SPE bit is reset).

Rx buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the Rx Buffer. It is reset when SPI Data register is read.

22.3.6 CRC calculation

A CRC calculator has been implemented for communication reliability. Separate CRC calculators are implemented for transmitted data and received data. The CRC is calculated using a programmable polynomial serially on each bit. It is calculated on the sampling clock edge defined by the CPHA and CPOL bits in the SPI_CR1 register.

Note: This SPI offers two kinds of CRC calculation standard which depend directly on the data frame format selected for the transmission and/or reception: 8-bit data (CR8) and 16-bit data (CRC16-CCITT).

CRC calculation is enabled by setting the CRCEN bit in the SPI_CR1 register. This action resets the CRC registers (SPI_RXCRCR and SPI_TXCRCR). When the CRCNEXT bit in SPI_CR1 is set, the SPI_TXCRCR value is transmitted at the end of the current byte transmission.

The CRCERR flag in the SPI_SR register is set if the value received in the shift register during the SPI_TXCRCR value transmission does not match the SPI_RXCRCR value.

If data are present in the TX buffer, the CRC value is transmitted only after the transmission of the data byte. During CRC transmission, the CRC calculator is switched off and the register value remains unchanged.

Note: Please refer to the product specifications for availability of this feature.

SPI communication using CRC is possible through the following procedure:

- Program the CPOL, CPHA, LSBFirst, BR, SSM, SSI and MSTR values
- Program the polynomial in the SPI_CRCPR register
- Enable the CRC calculation by setting the CRCEN bit in the SPI_CR1 register. This also clears the SPI_RXCRCR and SPI_TXCRCR registers
- Enable the SPI by setting the SPE bit in the SPI_CR1 register
- Start the communication and sustain the communication until all but one byte or half-word have been transmitted or received.
- On writing the last byte or half-word to the TX buffer, set the CRCNext bit in the SPI_CR1 register to indicate that after transmission of the last byte, the CRC should be transmitted. CRC calculation is frozen during the CRC transmission.
- After transmitting the last byte or half word, the SPI transmits the CRC. The CRCNEXT bit is reset. The CRC is also received and compared against the SPI_RXCRCR value. If the value does not match, the CRCERR flag in SPI_SR is set and an interrupt can be generated when the ERRIE bit in the SPI_CR2 register is set.

Note: With high bit rate frequencies, the user must be carefull when transmitting the CRC. As the number of used CPU cycles has to be as low as possible in the CRC transfer phase, the calling of software functions in the CRC transmission sequence is forbidden to avoid errors in the last data and CRC reception.

For high bit rate frequencies, it is advised to use the DMA mode to avoid the degradation of the SPI speed performance due to CPU accesses impacting the SPI bandwidth.

When the STM32F10xxx is configured as slave and the NSS hardware mode is used, the NSS pin needs to be kept low between the data phase and the CRC phase.

22.3.7 SPI communication using DMA (direct memory addressing)

To operate at its maximum speed, the SPI needs to be fed with the data for transmission and the data received on the Rx buffer should be read to avoid overrun. To facilitate the transfers, the SPI is implemented with a DMA facility with a simple request/acknowledge protocol. DMA access is requested when the enable bit in the SPI_CR2 register is enabled. There are separate requests for the Tx buffer and the Rx buffer.

DMA capability with CRC

When SPI communication is enabled with the CRC communication and the DMA mode, the transmission and reception of the CRC bytes at the end of communication are done automatically.

At the end of data and CRC transfers, the CRCERR flag in SPI_SR is set if corruption occurs during the transfer.

22.3.8 Error flags

Master mode fault (MODF)

Master mode fault occurs when the master device has its NSS pin pulled low (in hardware mode) or SSI bit low (in software mode), this automatically sets the MODF bit. Master mode fault affects the SPI peripheral in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is reset. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is reset, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1. Make a read or write access to the SPI_SR register while the MODF bit is set.
2. Then write to the SPI_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state during or after this clearing sequence.

As a security, hardware does not allow the setting of the SPE and MSTR bits while the MODF bit is set.

In a slave device the MODF bit cannot be set. However, in a multimaster configuration, the device can be in slave mode with this MODF bit set. In this case, the MODF bit indicates that there might have been a multimaster conflict for system control. An interrupt routine can be used to recover cleanly from this state by performing a reset or returning to a default state.

Overrun condition

An overrun condition occurs when the master device has sent data bytes and the slave device has not cleared the RXNE bit resulting from the previous data byte transmitted. When an overrun condition occurs:

- OVR bit is set and an interrupt is generated if the ERRIE bit is set.

In this case, the receiver buffer contents will not be updated with the newly received data from the master device. A read to the SPI_DR register returns this byte. All other subsequently transmitted bytes are lost.

Clearing the OVR bit is done by a read of the SPI_DR register followed by a read access to the SPI_SR register.

CRC error

This flag is used to verify the validity of the value received when the CRCEN bit in the SPI_CR1 register is set. The CRCERR flag in the SPI_SR register is set if the value

received in the shift register (after transmission of the transmitter SPI_TXCRCR value) does not match the receiver SPI_RXCRCR value.

22.3.9 Disabling the SPI

When transfer is terminated, the application can stop the communication by disabling the SPI peripheral. This is done by resetting the SPE bit. Disabling the SPI peripheral while the last data transfer is still ongoing does not affect the data reliability if the device is *not* in Master transmit mode.

Note: In Master transmit mode (full-duplex or simplex transmit only), the application must make sure that no data transfer is ongoing by checking the BSY flag in the SPI_SR register before disabling the SPI master.

22.3.10 SPI interrupts

Table 147. SPI interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
CRC error flag	CRCERR	

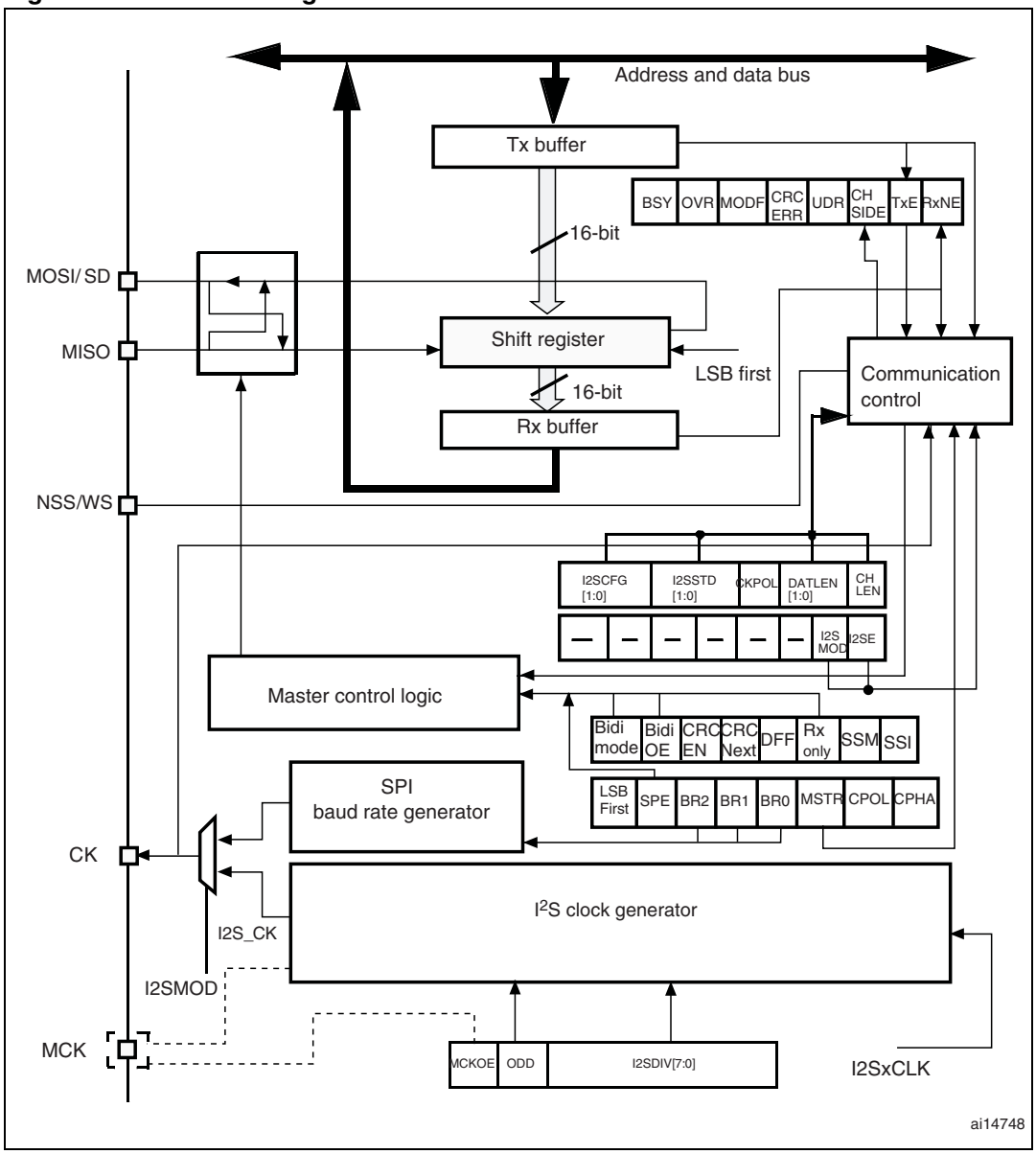
22.4 I^2S functional description

The I^2S audio protocol is not available in low- and medium-density devices. This section concerns only high-density devices.

22.4.1 General description

The block diagram of the I^2S is shown in *Figure 210*.

Figure 210. I²S block diagram



The SPI could function as an audio I²S interface when the I²S capability is enabled (by setting the I2SMOD bit in the SPI_I2SCFGR register). This interface uses almost the same pins, flags and interrupts as the SPI.

The I²S shares three common pins with the SPI:

- SD: Serial Data (mapped on the MOSI pin) to transmit or receive the two time-multiplexed data channels (in simplex mode only).
- WS: Word Select (mapped on the NSS pin) is the data control signal output in master mode and input in slave mode.
- CK: Serial Clock (mapped on the SCK pin) is the serial clock output in master mode and serial clock input in slave mode.

An additional pin could be used when a master clock output is needed for some external audio devices:

- MCK: Master Clock (mapped separately) is used, when the I^2S is configured in master mode (and when the MCKOE bit in the SPI_I2SPR register is set), to output this additional clock generated at a preconfigured frequency rate equal to $256 \times F_S$, where F_S is the audio sampling frequency.

The I^2S uses its own clock generator to produce the communication clock when it is set in master mode. This clock generator is also the source of the master clock output. Two additional registers are available in I^2S mode. One is linked to the clock generator configuration SPI_I2SPR and the other one is a generic I^2S configuration register SPI_I2SCFGR (audio standard, slave/master mode, data format, packet frame, clock polarity, etc.).

The SPI_CR1 register and all CRC registers are not used in the I^2S mode. Likewise, the SSOE bit in the SPI_CR2 register and the MODF and CRCERR bits in the SPI_SR are not used.

The I^2S uses the same SPI register for data transfer (SPI_DR) in 16-bit wide mode.

22.4.2 Supported audio protocols

The three-line bus has to handle only audio data generally time-multiplexed on two channels: the right channel and the left channel. However there is only one 16-bit register for the transmission or the reception. So, it is up to the software to write into the data register the adequate value corresponding to the considered channel side, or to read the data from the data register and to identify the corresponding channel by checking the CHSIDE bit in the SPI_SR register. Channel Left is always sent first followed by the channel right (CHSIDE has no meaning for the PCM protocol).

Four data and packet frames are available. Data may be sent with a format of:

- 16-bit data packed in 16-bit frame
- 16-bit data packed in 32-bit frame
- 24-bit data packed in 32-bit frame
- 32-bit data packed in 32-bit frame

When using 16-bit data extended on 32-bit packet, the first 16 bits (MSB) are the significant bits, the 16-bit LSB is forced to 0 without any need for software action or DMA request (only one read/write operation).

The 24-bit and 32-bit data frames need two CPU read or write operations to/from the SPI_DR or two DMA operations if the DMA is preferred for the application. For 24-bit data frame specifically, the 8 non-significant bits are extended to 32 bits with 0-bits (by hardware).

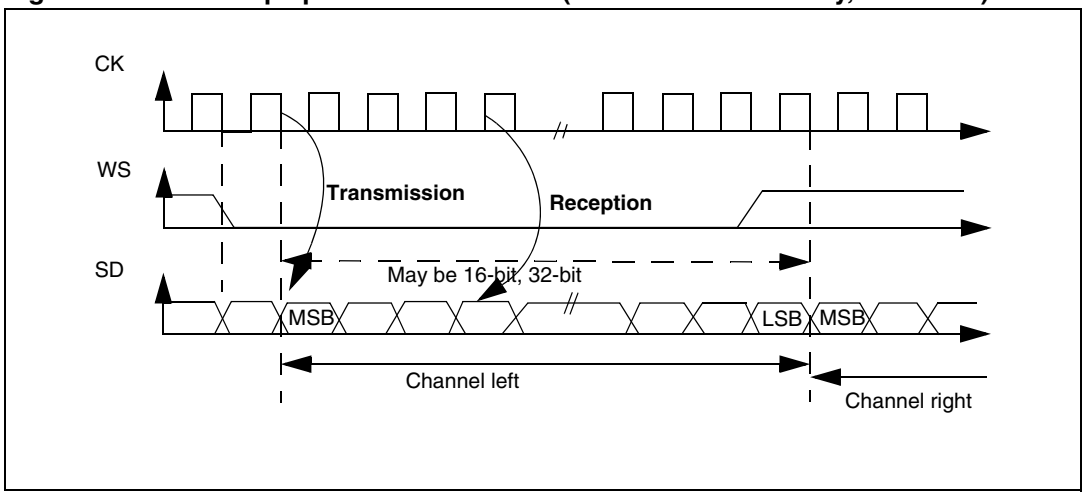
For all data formats and communication standards, the most significant bit is always sent first (MSB first).

The I^2S interface supports four audio standards, configurable using the I2SSTD[1:0] and PCMSYNC bits in the SPI_I2SCFGR register.

I^2S Phillips standard

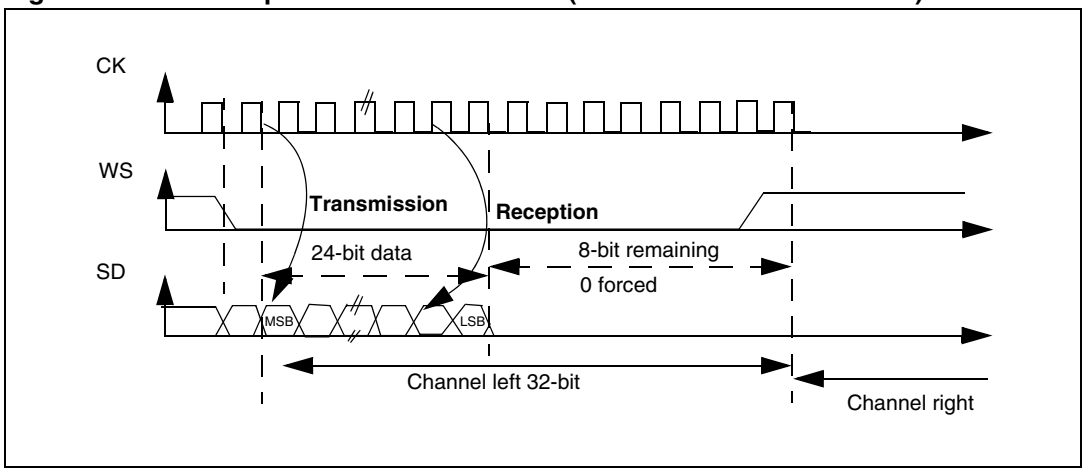
For this standard, the WS signal is used to indicate which channel is being transmitted. It is activated one CK clock cycle before the first bit (MSB) is available.

Figure 211. I²S Phillips protocol waveforms (16/32-bit full accuracy, CPOL = 0)



Data are latched on the falling edge of CK (for the transmitter) and are read on the rising edge (for the receiver). The WS signal is also latched on the falling edge of CK.

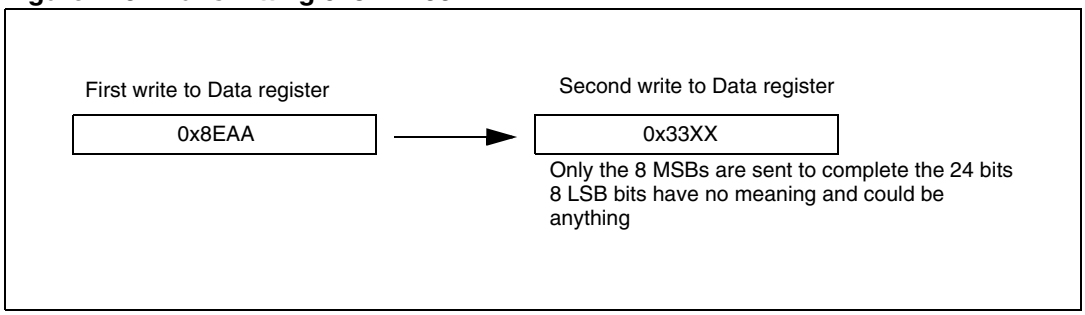
Figure 212. I²S Phillips standard waveforms (24-bit frame with CPOL = 0)



This mode needs two write or read operations to/from the SPI_DR.

- In transmission mode:
  if 0x8EAA33 has to be sent (24-bit):

Figure 213. Transmitting 0x8EAA33



- In reception mode:
  if data 0x8EAA33 is received:

Figure 214. Receiving 0x8EAA33

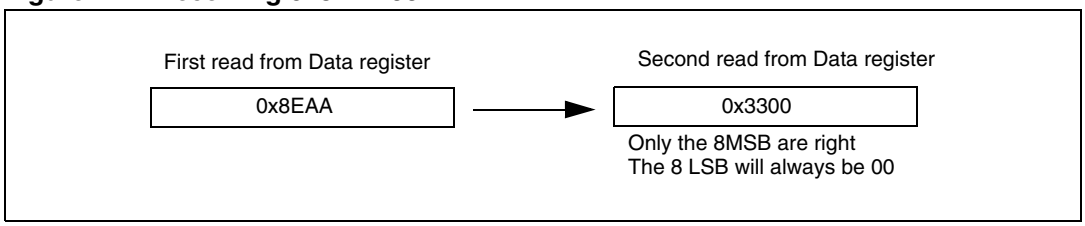
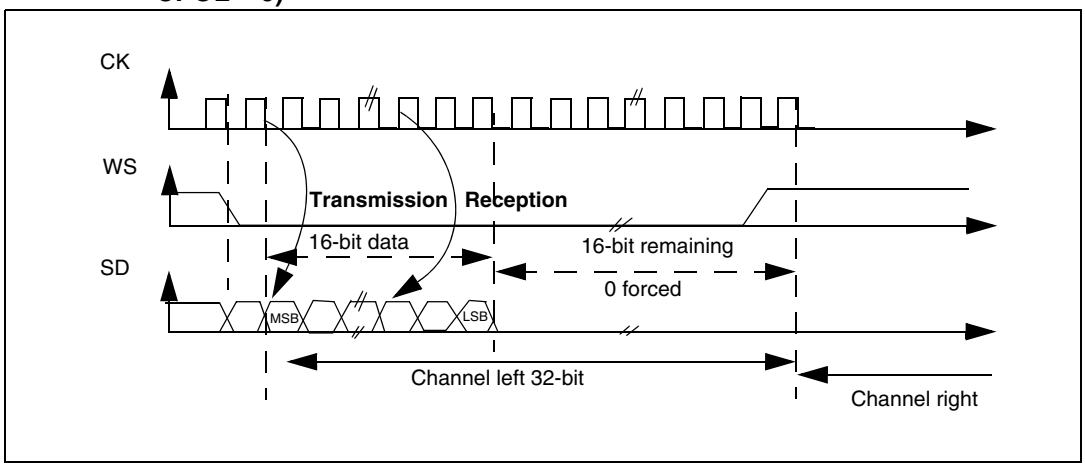


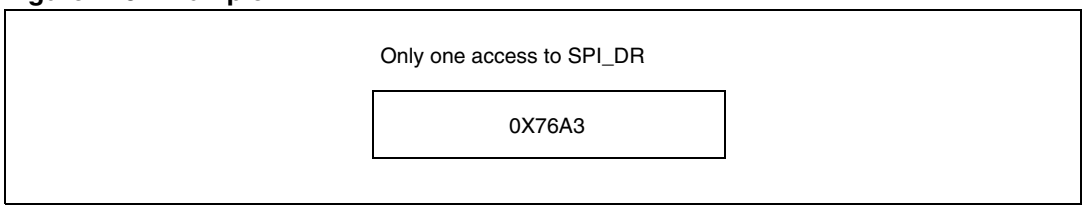
Figure 215. I²S Phillips standard (16-bit extended to 32-bit packet frame with CPOL = 0)



When 16-bit data frame extended to 32-bit channel frame is selected during the I²S configuration phase, only one access to SPI_DR is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format.

If the data to transmit or the received data are 0x76A3 (0x76A30000 extended to 32-bit), the operation shown in *Figure 216* is required.

Figure 216. Example



For transmission, each time an MSB is written to SPI_DR, the TXE flag is set and its interrupt, if allowed, is generated to load SPI_DR with the new value to send. This takes place even if 0x0000 have not yet been sent because it is done by hardware.

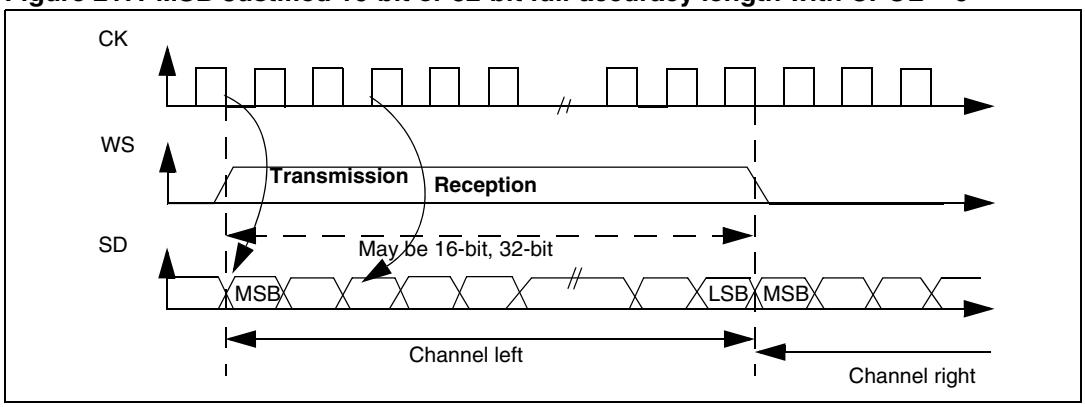
For reception, the RXNE flag is set and its interrupt, if allowed, is generated when the first 16 MSB half-word is received.

In this way, more time is provided between two write or read operations, which prevents underrun or overrun conditions (depending on the direction of the data transfer).

MSB justified standard

For this standard, the WS signal is generated at the same time as the first data bit, which is the MSBit.

Figure 217. MSB Justified 16-bit or 32-bit full-accuracy length with CPOL = 0



Data are latched on the falling edge of CK (for transmitter) and are read on the rising edge (for the receiver).

Figure 218. MSB Justified 24-bit frame length with CPOL = 0

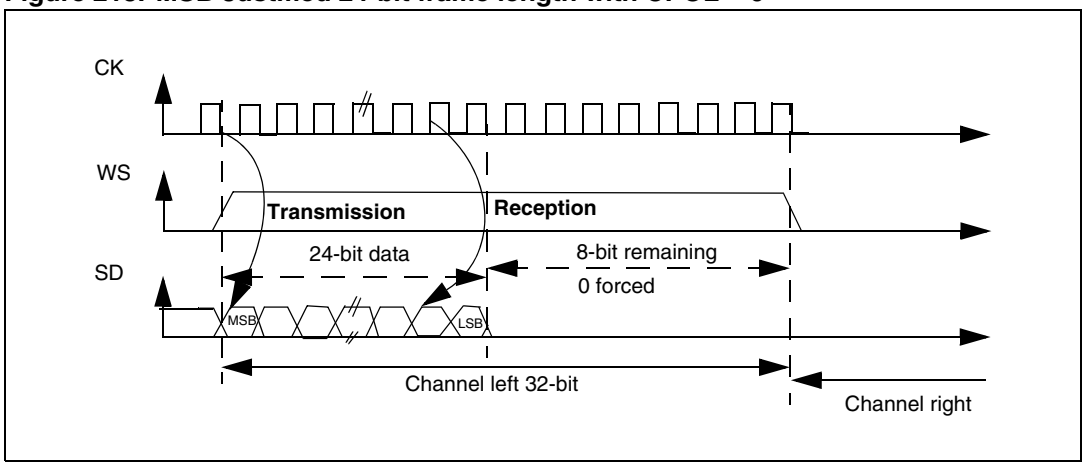
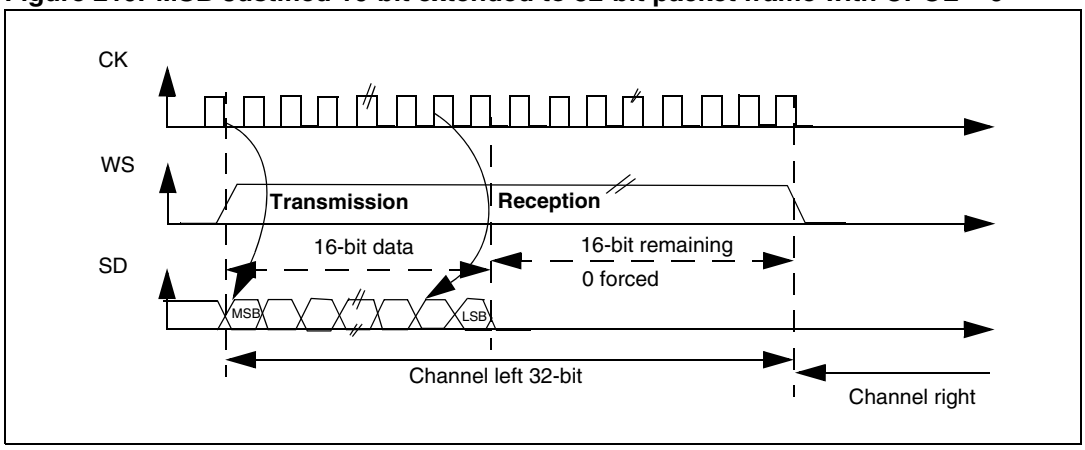


Figure 219. MSB Justified 16-bit extended to 32-bit packet frame with CPOL = 0



LSB justified standard

This standard is similar to the MSB justified standard (no difference for the 16-bit and 32-bit full-accuracy frame formats).

Figure 220. LSB justified 16-bit or 32-bit full-accuracy with CPOL = 0

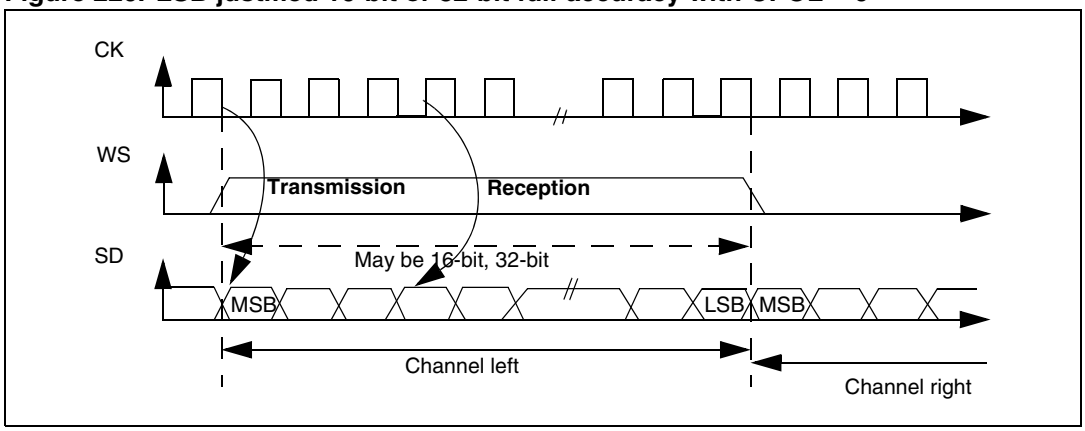
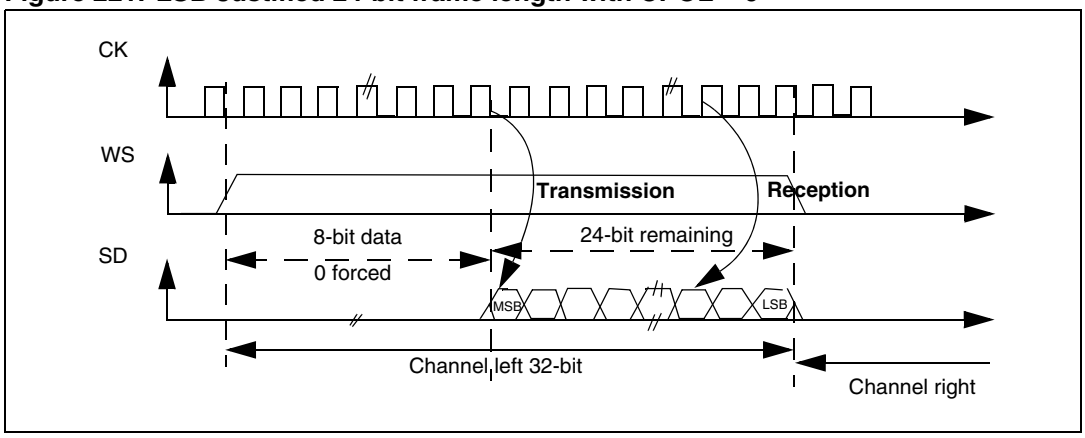


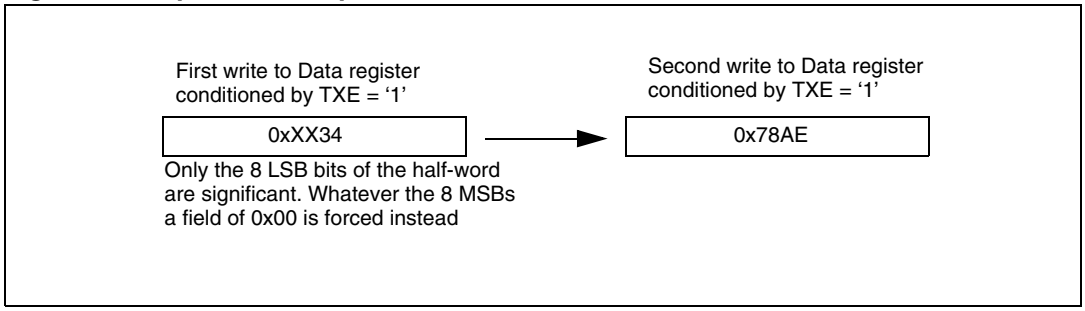
Figure 221. LSB Justified 24-bit frame length with CPOL = 0



- In transmission mode:

  If data 0x3478AE have to be transmitted, two write operations to the SPI_DR register are required from software or by DMA. The operations are shown below.

Figure 222. Operations required to transmit 0x3478AE



- In reception mode:

  If data 0x3478AE are received, two successive read operations from SPI_DR are required on each RXNE event.

Figure 223. Operations required to receive 0x3478AE

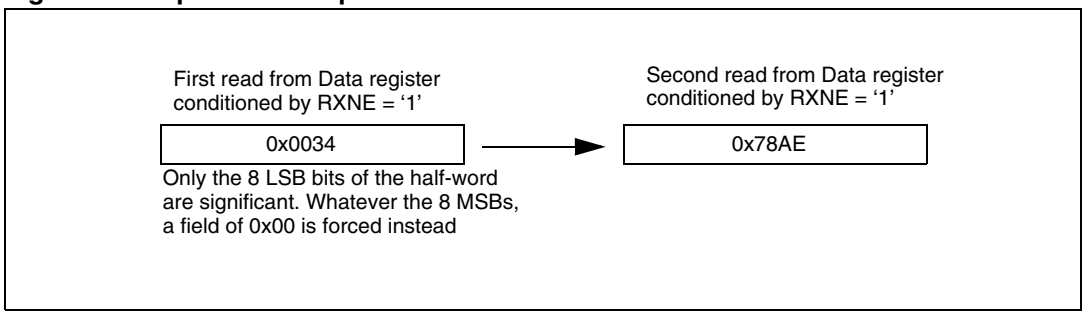
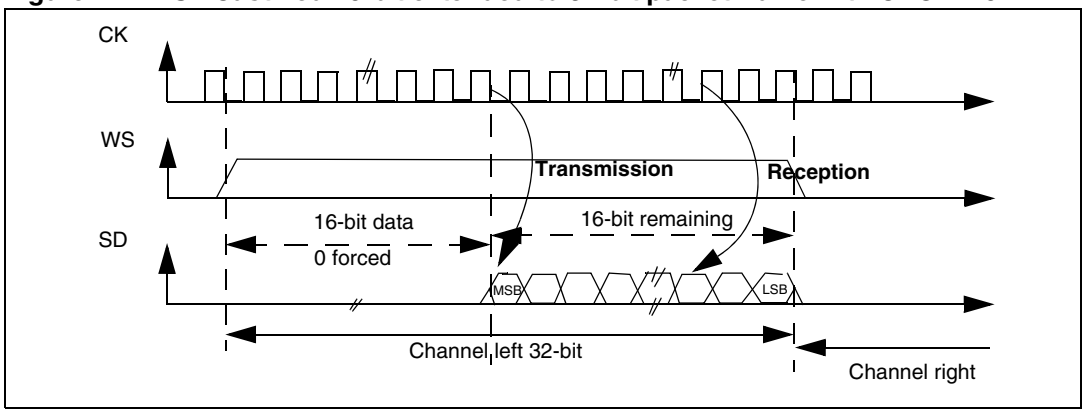


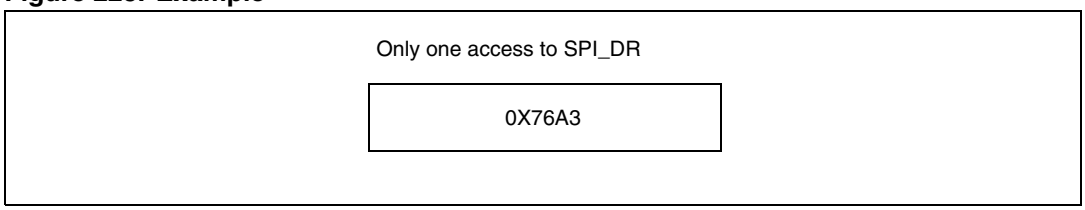
Figure 224. LSB Justified 16-bit extended to 32-bit packet frame with CPOL = 0



When 16-bit data frame extended to 32-bit channel frame is selected during the I^2S configuration phase, Only one access to SPI_DR is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format. In this case it corresponds to the half-word MSB.

If the data to transmit or the received data are 0x76A3 (0x0000 76A3 extended to 32-bit), the operation shown in *Figure 225* is required.

Figure 225. Example



In transmission mode, when TXE is asserted, the application has to write the data to be transmitted (in this case 0x76A3). The 0x000 field is transmitted first (extension on 32-bit). TXE is asserted again as soon as the effective data (0x76A3) is sent on SD.

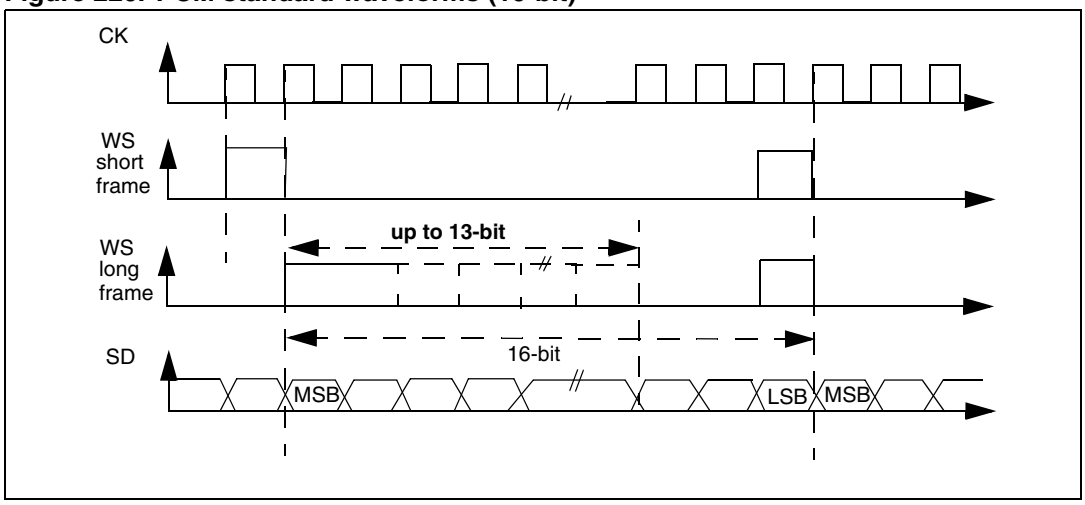
In reception mode, RXNE is asserted as soon as the significant half-word is received (and not the 0x0000 field).

In this way, more time is provided between two write or read operations to prevent underrun or overrun conditions.

PCM standard

For the PCM standard, there is no need to use channel-side information. The two PCM modes (short and long frame) are available and configurable using the PCMSYNC bit in SPI_I2SCFGR.

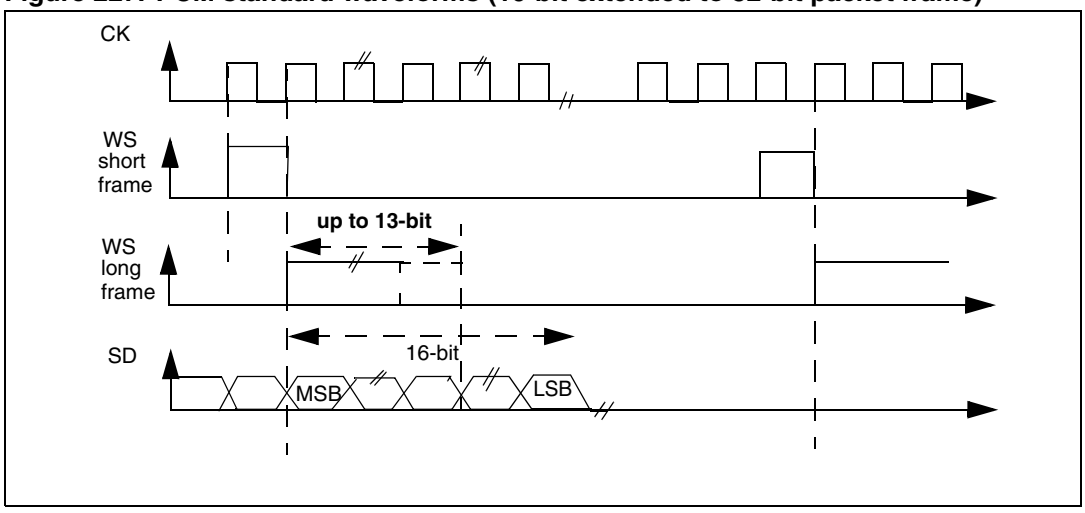
Figure 226. PCM standard waveforms (16-bit)



For long frame synchronization, the WS signal assertion time is fixed 13 bits in master mode.

For short frame synchronization, the WS synchronization signal is only one cycle long.

Figure 227. PCM standard waveforms (16-bit extended to 32-bit packet frame)



Note: For both modes (master and slave) and for both synchronizations (short and long), the number of bits between two consecutive pieces of data (and so two synchronization signals) needs to be specified (DATLEN and CHLEN bits in the SPI_I2SCFGR register) even in slave mode.

22.4.3 Clock generator

The I^2S bitrate determines the dataflow on the I^2S data line and the I^2S clock signal frequency.

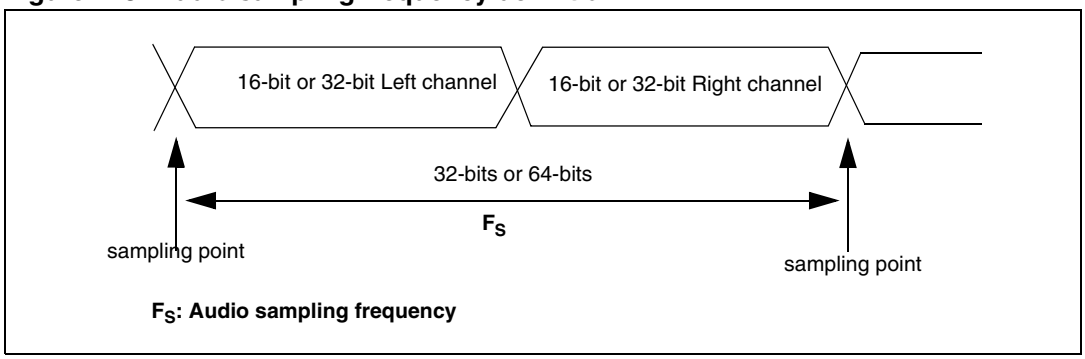
I^2S bitrate = number of bits per channel × number of channels × sampling audio frequency

For a 16-bit audio, left and right channel, the I^2S bitrate is calculated as follows:

I^2S bitrate = $16 \times 2 \times F_S$

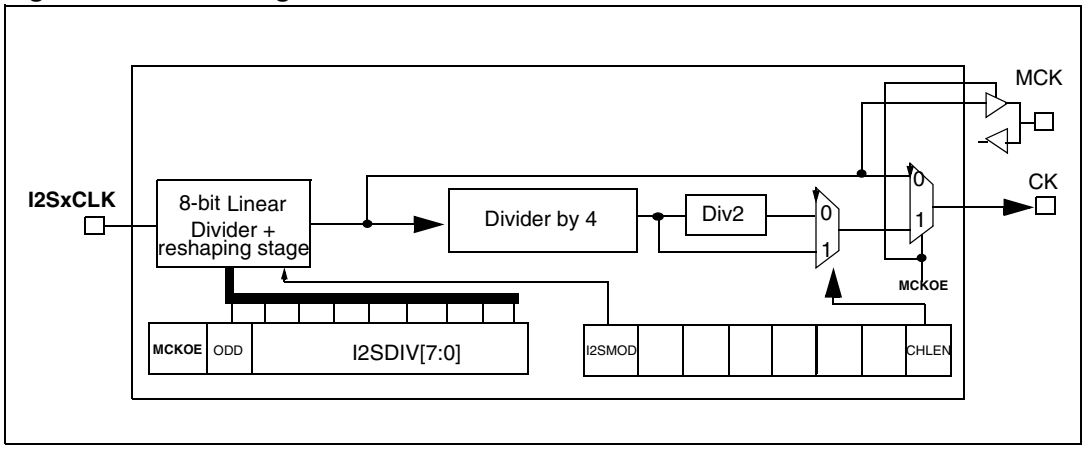
It will be: I^2S bitrate = 32 x 2 x F_S if the packet length is 32-bit wide.

Figure 228. Audio sampling frequency definition



When the master mode is configured, a specific action needs to be taken to properly program the linear divider in order to communicate with the desired audio frequency.

Figure 229. I^2S clock generator architecture



1. Where x could be 2 or 3.

Figure 228 presents the communication clock architecture. The I2SxCLK source is the System Clock (provided by the HSI, the HSE or the PLL and sourcing the AHB clock).

The audio sampling frequency may be 48 kHz, 44.1 kHz, 22.05 kHz, 16 kHz or 8 kHz. In order to reach the desired frequency, the linear divider needs to be programmed according to the formulas below:

When the master clock is generated (MCKOE in the SPI_I2SPR register is set):

F_S = I2SxCLK / [(16*2)*((2*I2SDIV)+ODD)*8)] when the channel frame is 16-bit wide

F_S = I2SxCLK / [(32*2)*((2*I2SDIV)+ODD)*4)] when the channel frame is 32-bit wide

When the master clock is disabled (MCKOE bit cleared):

F_S = I2SxCLK / [(16*2)*((2*I2SDIV)+ODD))] when the channel frame is 16-bit wide

F_S = I2SxCLK / [(32*2)*((2*I2SDIV)+ODD))] when the channel frame is 32-bit wide

22.4.4 I²S master mode

The I²S can be configured in master mode. This means that the serial clock is generated on the CK pin as well as the Word Select signal WS. Master clock (MCK) may be output or not, thanks to the MCKOE bit in the SPI_I2SPR register.

Procedure

1. Select the I2SDIV[7:0] bits in the SPI_I2SPR register to define the serial clock baud rate to reach the proper audio sample frequency. The ODD bit in the SPI_I2SPR register also has to be defined.
2. Select the CKPOL bit to define the steady level for the communication clock. Set the MCKOE bit in the SPI_I2SPR register if the master clock MCK needs to be provided to the external DAC/ADC audio component (the I2SDIV and ODD values should be computed depending on the state of the MCK output, for more details refer to *Section 22.4.3: Clock generator*).
3. Set the I2SMOD bit in SPI_I2SCFGR to activate the I²S functionalities and choose the I²S standard through the I2SSTD[1:0] and PCMSYNC bits, the data length through the DATLEN[1:0] bits and the number of bits per channel by configuring the CHLEN bit. Select also the I²S master mode and direction (Transmitter or Receiver) through the I2SCFG[1:0] bits in the SPI_I2SCFGR register.
4. If needed, select all the potential interruption sources and the DMA capabilities by writing the SPI_CR2 register.
5. The I2SE bit in SPI_I2SCFGR register must be set.

WS and CK are configured in output mode. MCK is also an output, if the MCKOE bit in SPI_I2SPR is set.

Transmission sequence

The transmission sequence begins when a half-word is written into the Tx buffer.

Assumedly, the first data written into the Tx buffer correspond to the channel Left data. When data are transferred from the Tx buffer to the shift register, TXE is set and data corresponding to the channel Right have to be written into the Tx buffer. The CHSIDE flag indicates which channel is to be transmitted. It has a meaning when the TXE flag is set because the CHSIDE flag is updated when TXE goes high.

A full frame has to be considered as a Left channel data transmission followed by a Right channel data transmission. It is not possible to have a partial frame where only the left channel is sent.

The data half-word is parallel loaded into the 16-bit shift register during the first bit transmission, and then shifted out, serially, to the MOSI/SD pin, MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

For more details about the write operations depending on the I²S standard mode selected, refer to *Section 22.4.2: Supported audio protocols*).

To ensure a continuous audio data transmission, it is mandatory to write the SPI_DR with the next data to transmit before the end of the current transmission.

To switch off the I^2S, by clearing I2SE, it is mandatory to wait for TXE = 0 and BSY = 0.

Reception sequence

The operating mode is the same as for the transmission mode except for the point 3, where the configuration should set the master reception mode through the I2SCFG[1:0] bits.

Whatever the data or channel length, the audio data are received by 16-bit packets. This means that each time the Rx buffer is full, the RXNE flag is set and an interrupt is generated if the RXNEIE bit is set in SPI_CR2 register. Depending on the data and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the Rx buffer.

Clearing the RXNE bit is performed by reading the SPI_DR register.

CHSIDE is updated after each reception. It is sensitive to the WS signal generated by the I^2S cell.

For more details about the read operations depending on the I^2S standard mode selected, refer to *Section 22.4.2: Supported audio protocols*.

If data are received while the precedent received data have not been read yet, an overrun is generated and the OVR flag is set. If the ERRIE bit is set in the SPI_CR2 register, an interrupt is generated to indicate the error.

To switch off the I^2S in reception mode, I2SE has to be cleared during and before the end of the last data reception. Even if I2SE is switched off while the last data are being transferred, the clock and the transfer are maintained until the end of the current data transmission.

22.4.5 I^2S slave mode

For the slave configuration, the I^2S can be configured in transmission or reception mode. The operating mode is following mainly the same rules as described for the I^2S master configuration. In slave mode, there is no clock to be generated by the I^2S interface. The clock and WS signals are input from the external master connected to the I^2S interface. There is then no need, for the user, to configure the clock.

The configuration steps to follow are listed below:

1. Set the I2SMOD bit in the SPI_I2SCFGR register to reach the I^2S functionalities and choose the I^2S standard through the I2SSTD[1:0] bits, the data length through the DATLEN[1:0] bits and the number of bits per channel for the frame configuring the CHLEN bit. Select also the mode (transmission or reception) for the slave through the I2SCFG[1:0] bits in SPI_I2SCFGR register.
2. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPI_CR2 register.
3. The I2SE bit in SPI_I2SCFGR register must be set.

Transmission sequence

The transmission sequence begins when a half-word (corresponding to channel Left data) is written to the Tx buffer. When data are transferred from the Tx buffer to the shift register, the TXE flag is set and data corresponding to the channel Right have to be written into the Tx buffer. The CHSIDE flag indicates which channel is to be transmitted. Compared to the

master transmission mode, in slave mode, CHSIDE is sensitive to the WS signal coming from the external master. This means that the slave needs to be ready to transmit the first data before the clock is generated by the master. WS assertion corresponds to channel Left transmitted first.

Note: The I2SE has to be written at least two PCLK cycles before the first clock of the master comes on the CK line.

The data half-word is parallel-loaded into the 16-bit shift register (from the internal bus) during the first bit transmission, and then shifted out serially to the MOSI/SD pin MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

Note that the TXE flag should be checked to be at 1 before attempting to write the Tx buffer.

For more details about the write operations depending on the I^2S standard mode selected, refer to *Section 22.4.2: Supported audio protocols*.

To secure a continuous audio data transmission, it is mandatory to write the SPI_DR register with the next data to transmit before the end of the current transmission. An underrun flag is set and an interrupt may be generated if the data are not written into the SPI_DR register before the first clock edge of the next data communication. This indicates to the software that the transferred data are wrong. If the ERRIE bit is set into the SPI_CR2 register, an interrupt is generated when the UDR flag in the SPI_SR register goes high. In this case, it is mandatory to switch off the I^2S and to restart a data transfer starting from the channel left.

Reception sequence

The operating mode is the same as for the transmission mode except for the point 1. where the configuration should set the master reception mode using the I2SCFG[1:0] bits in the SPI_I2SCFGR register.

Whatever the data length or the channel length, the audio data are received by 16-bit packets. This means that each time the RX buffer is full, the RXNE flag in the SPI_SR register is set and an interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register. Depending on the data length and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the RX buffer.

The CHSIDE flag is updated each time data are received to be read from SPI_DR. It is sensitive to the external WS line managed by the external master component.

Clearing the RXNE bit is performed by reading the SPI_DR register.

For more details about the read operations depending the I^2S standard mode selected, refer to *Section 22.4.2: Supported audio protocols*.

If data are received while the precedent received data have not yet been read, an overrun is generated and the OVR flag is set. If the bit ERRIE is set in the SPI_CR2 register, an interrupt is generated to indicate the error.

To switch off the I^2S in reception mode, I2SE has to be cleared during and before the end of the last data reception. Even if I2SE is switched off while the last data is being transferred, the clock and the transfer go on until the end of the last data transmission.

Note: The external master components should have the capability to send/receive data on 16-bit or 32-bit packet via an audio channel.

22.4.6 Status flags

Three status flags are provided for the application to fully monitor the state of the I^2S bus.

Busy flag (BSY)

This flag indicates the state of the I^2S communication layer. It is set to indicate that the I^2S is busy communicating and/or that there is a valid data half-word in the Tx buffer awaiting transmission. The purpose of this flag is to indicate if there is any communication ongoing on the I^2S bus or not. This flag becomes set as soon as:

1. Data are written into the SPI_DR register in master mode
2. The CK clock is present in slave mode

The Busy flag is reset as soon as a half-word is transmitted/received. It is set and cleared by hardware. This flag can be monitored to avoid write collision errors. Writing to it has no effect. It is meaningful only when the I2SE bit in the SPI_I2SCFGR register is set.

Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can then be loaded into it. The TXE flag is reset when the Tx buffer already contains data to be transmitted. It is also reset when the I^2S is disabled (I2SE bit is reset).

RX buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the RX Buffer. It is reset when SPI_DR register is read.

Channel Side flag (CHSIDE)

In transmission mode, this flag is refreshed when TXE goes high. It indicates the channel side to which the data to transfer on SD has to belong. In case of an underrun error event in slave transmission mode, this flag is not reliable and I^2S needs to be switched off and switched on before resuming the communication.

In reception mode, this flag is refreshed when data are received into SPI_DR. It indicates from which channel side data have been received. Note that in case of error (like OVR) this flag becomes meaningless and the I^2S should be reset by disabling and then enabling it (with configuration if it needs changing).

This flag has no meaning in the PCM standard (for both Short and Long frame modes).

When the OVR or UDR flag in the SPI_SR is set and the ERRIE bit in SPI_CR2 is also set, an interrupt is generated. This interrupt can be cleared by reading the SPI_SR status register (once the interrupt source has been cleared).

22.4.7 Error flags

There are two error flags for the I^2S cell.

Underrun flag (UDR)

In slave transmission mode this flag is set when the first clock for data transmission appears while the software has not yet loaded any value into SPI_DR. It is available when the I2SMOD bit in SPI_I2SCFGR is set. An interrupt may be generated if the ERRIE bit in SPI_CR2 is set.
The UDR bit is cleared by a read operation on the SPI_SR register.

Overrun flag (OVR)

This flag is set when data are received and the previous data have not yet been read from SPI_DR. As a result, the incoming data are lost. An interrupt may be generated if the ERRIE bit is set in SPI_CR2.

In this case, the receive buffer contents are not updated with the newly received data from the transmitter device. A read operation to the SPI_DR register returns the previous correctly received data. All other subsequently transmitted half-words are lost.

Clearing the OVR bit is done by a read operation on the SPI_DR register followed by a read access to the SPI_SR register.

22.4.8 I²S interrupts

Table 148 provides the list of I²S interrupts.

Table 148. I²S interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Overrun error	OVR	ERRIE
Underrun error	UDR	

22.4.9 DMA features

DMA is working in exactly the same way as for the SPI mode. There is no difference on the I²S. Only the CRC feature is not available in I²S mode since there is no data transfer protection system.

22.5 SPI and I²S registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

22.5.1 SPI Control Register 1 (SPI_CR1) (not used in I²S mode)

Address offset: 0x00

Reset value: 0x0000)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **BIDIMODE:** *Bidirectional data mode enable*

0: 2-line unidirectional data mode selected

1: 1-line bidirectional data mode selected

Note: Not used in I2S mode

Bit 14 **BIDIOE:** *Output enable in bidirectional mode*

This bit combined with the BIDImode bit selects the direction of transfer in bidirectional mode

0: Output disabled (receive-only mode)

1: Output enabled (transmit-only mode)

Notes: In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.

Not used in I^2S mode

Bit 13 **CRCEN:** *Hardware CRC calculation enable*

0: CRC calculation disabled

1: CRC calculation Enabled

Notes: This bit should be written only when SPI is disabled (SPE = ‘0’) for correct operation

Not used in I^2S mode

Bit 12 **CRCNEXT:** *Transmit CRC next*

0: Next transmit value is from Tx buffer

1: Next transmit value is from Tx CRC register

Notes: This bit has to be written as soon as the last data is written into the SPI_DR register.

Not used in I^2S mode

Bit 11 **DFF:** *Data Frame Format*

0: 8-bit data frame format is selected for transmission/reception

1: 16-bit data frame format is selected for transmission/reception

Notes: This bit should be written only when SPI is disabled (SPE = ‘0’) for correct operation

Not used in I2S mode

Bit 10 **RXONLY:** *Receive only*

This bit combined with the BIDImode bit selects the direction of transfer in 2-line unidirectional mode. This bit is also useful in a multislave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

0: Full duplex (Transmit and receive)

1: Output disabled (Receive-only mode)

Note: Not used in I^2S mode

Bit 9 **SSM:** *Software slave management*

When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.

0: Software slave management disabled

1: Software slave management enabled

Note: Not used in I^2S mode

Bit 8 **SSI:** *Internal slave select*

This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the I/O value of the NSS pin is ignored.

Note: Not used in I^2S mode

Bit 7 **LSBFIRST:** *Frame Format*

0: MSB transmitted first

1: LSB transmitted first

Notes: This bit should not be changed when communication is ongoing.

Not used in I^2S mode

Bit 6 **SPE:** *SPI Enable*

0: Peripheral disabled

1: Peripheral enabled

Note: Not used in I^2S mode

Bits 5:3 **BR[2:0]:** *Baud Rate Control*

000: $f_{PCLK}/2$

001: $f_{PCLK}/4$

010: $f_{PCLK}/8$

011: $f_{PCLK}/16$

100: $f_{PCLK}/32$

101: $f_{PCLK}/64$

110: $f_{PCLK}/128$

111: $f_{PCLK}/256$

Notes: These bits should not be changed when communication is ongoing.

Not used in I^2S mode

Bit 2 **MSTR:** *Master Selection*

0: Slave configuration

1: Master configuration

Notes: This bit should not be changed when communication is ongoing.

Not used in I^2S mode

Bit1 **CPOL:** *Clock Polarity*

0: CK to 0 when idle

1: CK to 1 when idle

Notes: This bit should not be changed when communication is ongoing.

Not used in I^2S mode

Bit 0 **CPHA:** *Clock Phase*

0: The first clock transition is the first data capture edge

1: The second clock transition is the first data capture edge

Notes: This bit should not be changed when communication is ongoing.

Not used in I^2S mode

22.5.2 SPI control register 2 (SPI_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	reserved		SSOE	TXDMAEN	RXDMAEN
Res.								rw	rw	rw	Res.		rw	rw	rw

Bits 15:8 Reserved. Forced to 0 by hardware.

Bit 7 **TXEIE:** *Tx buffer Empty Interrupt Enable*

0: TXE interrupt masked

1: TXE interrupt not masked. Used to generate an interrupt request when the TXE flag is set.

Note: To function correctly, the TXEIE and TXDMAEN bits should not be set at the same time.

Bit 6 **RXNEIE:** *RX buffer Not Empty Interrupt Enable*

0: RXNE interrupt masked

1: RXNE interrupt not masked. Used to generate an interrupt request when the RXNE flag is set.

Note: To function correctly, the RXNEIE and RXDMAEN bits should not be set at the same time.

Bit 5 **ERRIE:** *Error Interrupt Enable*

This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode and UDR, OVR in I^2S mode).

0: Error interrupt is masked

1: Error interrupt is enabled.

Bits 4:3 Reserved. Forced to 0 by hardware.

Bit 2 **SSOE:** *SS Output Enable*

0: SS output is disabled in master mode and the cell can work in multimaster configuration

1: SS output is enabled in master mode and when the cell is enabled. The cell cannot work in a multimaster environment.

Note: Not used in I^2S mode

Bit 1 **TXDMAEN:** *Tx Buffer DMA Enable*

When this bit is set, the DMA request is made whenever the TXE flag is set.

0: Tx buffer DMA disabled

1: Tx buffer DMA enabled

Bit 0 **RXDMAEN:** *Rx Buffer DMA Enable*

When this bit is set, the DMA request is made whenever the RXNE flag is set.

0: Rx buffer DMA disabled

1: Rx buffer DMA enabled

22.5.3 SPI status register (SPI_SR)

Address offset: 08h

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
Res.								r	r	r	rc_w0	r	r	r	r

Bits 15:8 Reserved. Forced to 0 by hardware.

Bit 7 **BSY:** Busy flag

0: SPI (or I2S) not busy

1: SPI (or I2S) is busy in communication or Tx buffer is not empty

This flag is set and cleared by hardware.

Note: In master receiver-only mode (1-line bidirectional), it is forbidden to check the BSY flag.

Bit 6 **OVR:** Overrun flag

0: No overrun occurred

1: Overrun occurred

This flag is set by hardware and reset by a software sequence. Refer to *Section 22.4.7 on page 556* for the software sequence.

Bit 5 **MODF:** Mode fault

0: No mode fault occurred

1: Mode fault occurred

This flag is set by hardware and reset by a software sequence. Refer to *Section 22.3.8 on page 542* for the software sequence.

Note: Not used in I^2S mode

Bit 4 **CRCERR:** CRC Error flag

0: CRC value received matches the SPI_RXCRCR value

1: CRC value received does not match the SPI_RXCRCR value

This flag is set by hardware and cleared by software writing 0.

Note: Not used in I^2S mode

Bit 3 **UDR:** Underrun flag

0: No underrun occurred

1: Underrun occurred

This flag is set by hardware and reset by a software sequence. Refer to *Section 22.4.7 on page 556* for the software sequence.

Note: Not used in SPI mode

Bit 2 **CHSIDE**: Channel side

0: Channel Left has to be transmitted or has been received

1: Channel Right has to be transmitted or has been received

Note: Not used for the SPI mode

Note: No meaning in PCM mode

Bit 1 **TXE:** Transmit buffer empty

0: Tx buffer not empty

1: Tx buffer empty

Bit 0 **RXNE:** Receive buffer not empty

0: Rx buffer empty

1: Rx buffer not empty

22.5.4 SPI data register (SPI_DR)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **DR[15:0]:** *Data Register*

Data received or to be transmitted.

The data register is split into 2 buffers - one for writing (Transmit Buffer) and another one for reading (Receive buffer). A write to the data register will write into the Tx buffer and a read from the data register will return the value held in the Rx buffer.

Notes for the SPI mode:

Depending on the data frame format selection bit (DFF in SPI_CR1 register), the data sent or received is either 8-bit or 16-bit. This selection has to be made before enabling the SPI to ensure correct operation.

For an 8-bit data frame, the buffers are 8-bit and only the LSB of the register (SPI_DR[7:0]) is used for transmission/reception. When in reception mode, the MSB of the register (SPI_DR[15:8]) is forced to 0.

For a 16-bit data frame, the buffers are 16-bit and the entire register, SPI_DR[15:0] is used for transmission/reception.

22.5.5 SPI CRC polynomial register (SPI_CRCPR) (not used in I²S mode)

Address offset: 0x10

Reset value: 0x0007

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRCPOLY[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CRCPOLY[15:0]:** *CRC polynomial register*

This register contains the polynomial for the CRC calculation.

The CRC polynomial (0007h) is the reset value of this register. Another polynomial can be configured as required.

Note: Not used for the I²S mode.

22.5.6 SPI Rx CRC register (SPI_RXCRCR) (not used in I²S mode)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RxCRC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **RXCRC[15:0]:** *Rx CRC Register*

When CRC calculation is enabled, the RxCRC[15:0] bits contain the computed CRC value of the subsequently received bytes. This register is reset when the CRCEN bit in SPI_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI_CRCPR register.

Only the 8 LSB bits are considered when the data frame format is set to be 8-bit data (DFF bit of SPI_CR1 is cleared). CRC calculation is done based on CRC8.

The entire 16-bits of this register are considered when a 16-bit data frame format is selected (DFF bit of the SPI_CR1 register is set). CRC calculation is done based on CRC16 - CCITT standard.

Note: A read to this register when the BSY Flag is set could return an incorrect value.
Not used for the I²S mode.

22.5.7 SPI Tx CRC register (SPI_TXCRCR) (not used in I²S mode)

Address offset: 0x18

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TxCRC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **TxCRC[15:0]:** *Tx CRC register*

When CRC calculation is enabled, the TxCRC[7:0] bits contain the computed CRC value of the subsequently transmitted bytes. This register is reset when the CRCEN bit of SPI_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI_CRCPR register.

Only the 8 LSB bits are considered when the data frame format is set to be 8-bit data (DFF bit of SPI_CR1 is cleared). CRC calculation is done based on CRC8.

The entire 16-bits of this register are considered when a 16-bit data frame format is selected (DFF bit of the SPI_CR1 register is set). CRC calculation is done based on CRC16 - CCITT standard.

Note: A read to this register when the BSY flag is set could return an incorrect value.
Not used for the I²S mode.

22.5.8 SPI_I²S configuration register (SPI_I2SCFGR)

Address offset: 1Ch

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				I2SMOD	I2SE	I2SCFG		PCMSYNC	reserved	I2SSTD		CKPOL	DATLEN		CHLEN
				rw		rw		rw		rw		rw	rw		rw

Bits 15:12 Reserved: Forced to 0 by hardware

Bit 11 **I2SMOD**: *I²S mode selection*

0: SPI mode is selected
1: I2S mode is selected

Note: This bit should be configured when the SPI or I²S is disabled

Bit 10 **I2SE**: *I²S Enable*

0: I²S peripheral is disabled
1: I²S peripheral is enabled

Note: Not used in SPI mode

Bit 9:8 **I2SCFG**: *I²S configuration mode*

00: Slave - transmit
01: Slave - receive
10: Master - transmit
11: Master - receive

Notes: This bit should be configured when the I²S is disabled.
Not used for the SPI mode

Bit 7 **PCMSYNC**: *PCM frame synchronization*

0: Short frame synchronization

1: Long frame synchronization

Notes: This bit has a meaning only if I2SSTD = 11 (PCM standard is used)

Not used for the SPI mode

Bit 6 Reserved: forced at 0 by hardware

Bit 5:4 **I2SSTD**: *I^2S standard selection*

00: I^2S Phillips standard.

01: MSB justified standard (left justified)

10: LSB justified standard (right justified)

11: PCM standard

For more details on I^2S standards, refer to *Section 22.4.2 on page 545*

Notes: For correct operation, these bits should be configured when the I^2S is disabled.

Not used in SPI mode

Bit 3 **CKPOL**: *steady state clock polarity*

0: I^2S clock steady state is low level

1: I^2S clock steady state is high level

Notes: For correct operation, this bit should be configured when the I^2S is disabled.

Not used in SPI mode

Bit 2:1 **DATLEN**: *Data length to be transferred*

00: 16-bit data length

01: 24-bit data length

10: 32-bit data length

11: Not allowed

Notes: For correct operation, these bits should be configured when the I^2S is disabled.

Not used in SPI mode

Bit 0 **CHLEN**: *Channel length (number of bits per audio channel)*

0: 16-bit wide

1: 32-bit wide

The bit write operation has a meaning only if DATLEN = 00 otherwise the channel length is fixed to 32-bit by hardware whatever the value filled in.

Notes: For correct operation, this bit should be configured when the I^2S is disabled.

Not used in SPI mode

22.5.9 SPI_I²S Prescaler register (SPI_I2SPR)

Address offset: 20h

Reset value: 0000 0010 (0002h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						MCKOE	ODD	I2SDIV							
Res.						rw	rw	rw							

Bits 15:10 Reserved: Forced to 0 by hardware

Bit 9 **MCKOE**: *Master Clock Output Enable*

0: Master clock output is disabled

1: Master clock output is enabled

Notes: This bit should be configured when the I²S is disabled. It is used only when the I²S is in master mode.

Not used in SPI mode.

Bit 8 **ODD**: *Odd factor for the prescaler*

0: real divider value is = I2SDIV *2

1: real divider value is = (I2SDIV * 2)+1

Refer to *Section 22.4.3 on page 552*

Notes: This bit should be configured when the I²S is disabled. It is used only when the I²S is in master mode.

Not used in SPI mode

Bit 7:0 **I2SDIV**: *I²S Linear prescaler*

I2SDIV [7:0] = 0 or I2SDIV [7:0] = 1 are forbidden values.

Refer to *Section 22.4.3 on page 552*

Notes: These bits should be configured when the I²S is disabled. It is used only when the I²S is in master mode.

Not used in SPI mode.

22.5.10 SPI register map

The table provides shows the SPI register map and reset values.

Table 149. SPI register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**SPI_CR1**	Reserved																BIDIMODE	BIDIOE	CRCEN	CRCNEXT	DFF	RXONLY	SSM	SSI	LSBFIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	**SPI_CR2**	Reserved																								TXEIE	RXNEIE	ERRIE	Reserved		SSOE	TXDMAEN	RXDMAEN
	Reset Value																									0	0	0			0	0	0
0x08	**SPI_SR**	Reserved																								BSY	OVR	MODF	CRCERR	UDR	CHSIDE	TXE	RXNE
	Reset Value																									0	0	0	0	0	0	1	0
0x0C	**SPI_DR**	Reserved																DR[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**SPI_CRCPR**	Reserved																CRCPOLY[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0x14	**SPI_RXCRCR**	Reserved																RxCRC[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**SPI_TXCRCR**	Reserved																TxCRC[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**SPI_I2SCFGR**	Reserved																				I2SMOD	I2SE	I2SCFG		PCMSYNC	Reserved	I2SSTD		CKPOL	DATLEN		CHLEN
	Reset Value																					0	0	0	0	0		0	0	0	0	0	0
0x20	**SPI_I2SPR**	Reserved																						MCKOE	ODD	I2SDIV							
	Reset Value																							0	0	0	0	0	0	0	0	1	0

Note: Refer to Table 1 on page 35 for the register boundary addresses.

23 Inter-integrated circuit (I^2C) interface

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This section applies to the whole STM32F10xxx family, unless otherwise specified.

23.1 I^2C Introduction

I^2C (inter-integrated circuit) bus Interface serves as an interface between the microcontroller and the serial I^2C bus. It provides multimaster capability, and controls all I^2C bus-specific sequencing, protocol, arbitration and timing. It supports standard and fast speed modes. It is also SMBus 2.0 compatible.

It may be used for a variety of purposes, including CRC generation and verification, SMBus (system management bus) and PMBus (power management bus).

Depending on specific device implementation DMA capability can be available for reduced CPU overload.

23.2 I^2C main features

- Parallel-bus/I^2C protocol converter
- Multimaster capability: the same interface can act as Master or Slave
- I^2C Master features:
  - Clock generation
  - Start and Stop generation
- I^2C Slave features:
  - Programmable I^2C Address detection
  - Dual Addressing Capability to acknowledge 2 slave addresses
  - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and General Call
- Supports different communication speeds:
  - Standard Speed (up to 100 kHz),
  - Fast Speed (up to 400 kHz)
- Status flags:
  - Transmitter/Receiver mode flag
  - End-of-Byte transmission flag
  - I^2C busy flag
- Error flags:
  - Arbitration lost condition for master mode

  - Acknowledgement failure after address/ data transmission
  - Detection of misplaced start or stop condition
  - Overrun/Underrun if clock stretching is disabled
- 2 Interrupt vectors:
  - 1 Interrupt for successful address/ data communication
  - 1 Interrupt for error condition
- Optional Clock Stretching
- 1-byte buffer with DMA capability
- Configurable PEC (Packet Error Checking) Generation or Verification:
  - PEC value can be transmitted as last byte in Tx mode
  - PEC error checking for last received byte
- SMBus 2.0 Compatibility:
  - 25 ms clock low timeout delay
  - 10 ms master cumulative clock low extend time
  - 25 ms slave cumulative clock low extend time
  - Hardware PEC generation/verification with ACK control
  - Address Resolution Protocol (ARP) supported
- PMBus Compatibility

Note: Some of the above features may not be available in certain products. The user should refer to the product data sheet, to identify the specific features supported by the I^2C interface implementation.

23.3 I^2C functional description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I^2C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz) or fast (up to 400 kHz) I^2C bus.

23.3.1 Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

Communication flow

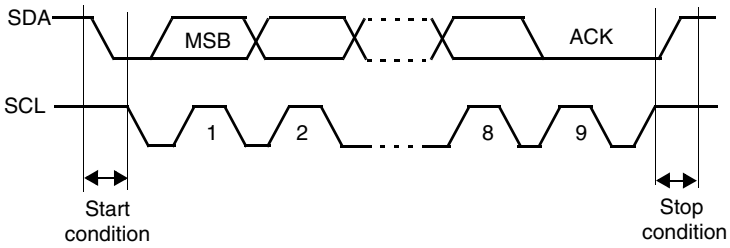
In Master mode, the I^2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to the following figure.

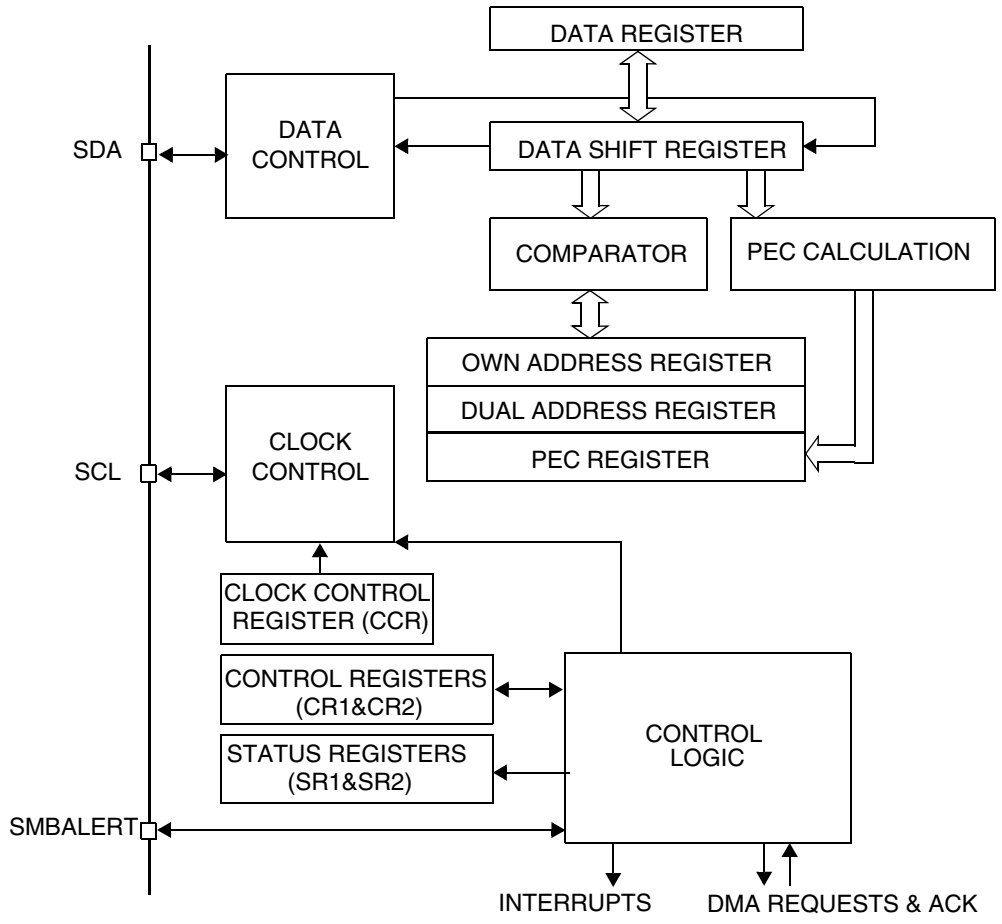
Figure 230. I^2C bus protocol



Acknowledge may be enabled or disabled by software. The I^2C interface addresses (dual addressing 7-bit/ 10-bit and/or general call address) can be selected by software.

The block diagram of the I^2C interface is shown in *Figure 231*.

Figure 231. I²C block diagram



Note: *SMBALERT is an optional signal in SMBus mode. This signal is not applicable if SMBus is disabled.*

23.3.2 I²C slave mode

By default the I²C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

The peripheral input clock must be programmed in the I2C_CR2 register in order to generate correct timings. The peripheral input clock frequency must be at least:

- 2 MHz in Standard mode
- 4 MHz in Fast mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register. Then it is compared with the address of the interface (OAR1) and with OAR2 (if ENDUAL=1) or the General Call address (if ENGC = 1).

Note: In 10-bit addressing mode, the comparison includes the header sequence (11110xx0), where xx denotes the two most significant bits of the address.

Header or address not matched: the interface ignores it and waits for another Start condition.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set and waits for the 8-bit slave address.

Address matched: the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.
- If ENDUAL=1, the software has to read the DUALF bit to check which slave address has been acknowledged.

In 10-bit mode, after receiving the address sequence the slave is always in Receiver mode. It will enter Transmitter mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the slave is in Receiver or Transmitter mode.

Slave transmitter

Following the address reception and after clearing ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

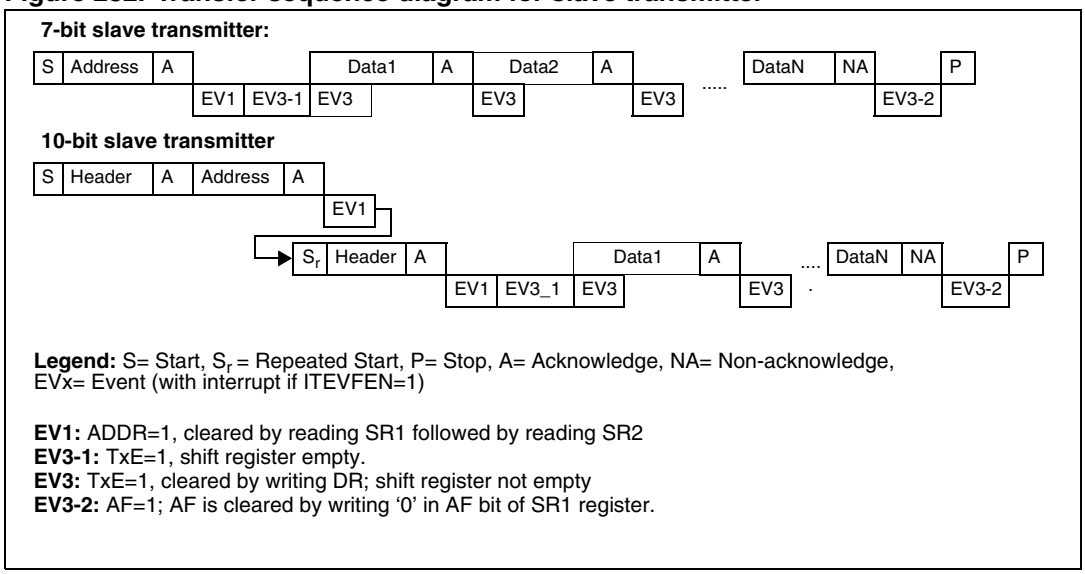
The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see *Figure 232* Transfer sequencing EV1 EV3).

When the acknowledge pulse is received:

- The TxE bit is set by hardware with an interrupt if the ITEVFEN and the ITBUFEN bits are set.

If TxE is set and a data was not written in the DR register before the end of the last data transmission, the BTF bit is set and the interface waits for a write in the DR register, stretching SCL low.

Figure 232. Transfer sequence diagram for slave transmitter



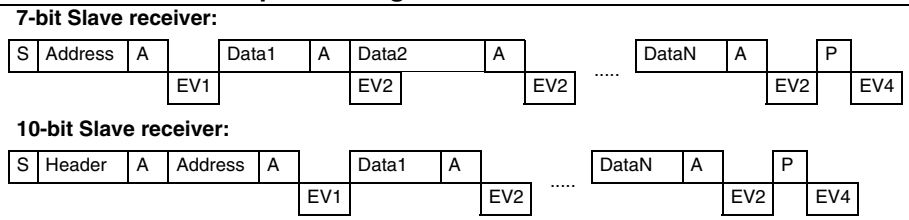
Slave receiver

Following the address reception and after clearing ADDR, the slave receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RxNE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bit is set.

If RxNE is set and the data in the DR register is not read before the end of the last data reception, the BTF bit is set and the interface waits for a read to the DR register, stretching SCL low (see *Figure 233* Transfer sequencing).

Figure 233. Transfer sequence diagram for slave receiver



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge, EVx= Event (with interrupt if ITEVFEN=1)

EV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV2: RxNE=1 cleared by reading DR register.
EV4: STOPF=1, cleared by reading SR1 register followed by writing to the CR1 register

Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets,

- The STOPF bit and generates an interrupt if the ITEVFEN bit is set.

Then the interface waits for a read of the SR1 register followed by a write to the CR1 register (see *Figure 233* Transfer sequencing EV4).

23.3.3 I^2C master mode

In Master mode, the I^2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a Start condition and ends with a Stop condition. Master mode is selected as soon as the Start condition is generated on the bus with a START bit.

The following is the required sequence in master mode.

- Program the peripheral input clock in I2C_CR2 Register in order to generate correct timings
- Configure the clock control registers
- Configure the rise time register
- Program the I2C_CR1 register to enable the peripheral
- Set the START bit in the I2C_CR1 register to generate a Start condition

The peripheral input clock frequency must be at least:

- 2 MHz in Standard mode
- 4 MHz in Fast mode

Start condition

Setting the START bit causes the interface to generate a Start condition and to switch to Master mode (M/SL bit set) when the BUSY bit is cleared.

Note: In master mode, setting the START bit causes the interface to generate a ReStart condition at the end of the current byte transfer.

Once the Start condition is sent:

- The SB bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the Slave address (see *Figure 234* & *Figure 235* Transfer sequencing EV5).

Slave address transmission

Then the slave address is sent to the SDA line via the internal shift register.

- In 10-bit addressing mode, sending the header sequence causes the following event:
  - The ADD10 bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

  Then the master waits for a read of the SR1 register followed by a write in the DR register with the second address byte (see *Figure 234* & *Figure 235* Transfer sequencing).
  - The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

  Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see *Figure 234* & *Figure 235* Transfer sequencing).
- In 7-bit addressing mode, one address byte is sent.

  As soon as the address byte is sent,
  - The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

  Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see *Figure 234* & *Figure 235* Transfer sequencing).

The master can decide to enter Transmitter or Receiver mode depending on the LSB of the slave address sent.

- In 7-bit addressing mode,
  - To enter Transmitter mode, a master sends the slave address with LSB reset.
  - To enter Receiver mode, a master sends the slave address with LSB set.
- In 10-bit addressing mode,
  - To enter Transmitter mode, a master sends the header (11110xx0) and then the slave address with LSB reset, (where xx denotes the two most significant bits of the address).
  - To enter Receiver mode, a master sends the header (11110xx0) and then the slave address with LSB reset. Then it should send a repeated Start condition followed by the header (11110xx1), (where xx denotes the two most significant bits of the address).

  The TRA bit indicates whether the master is in Receiver or Transmitter mode.

Master transmitter

Following the address transmission and after clearing ADDR, the master sends bytes from the DR register to the SDA line via the internal shift register.

The master waits until TxE is cleared, (see *Figure 234* Transfer sequencing EV8).

When the acknowledge pulse is received:

- The TxE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set.

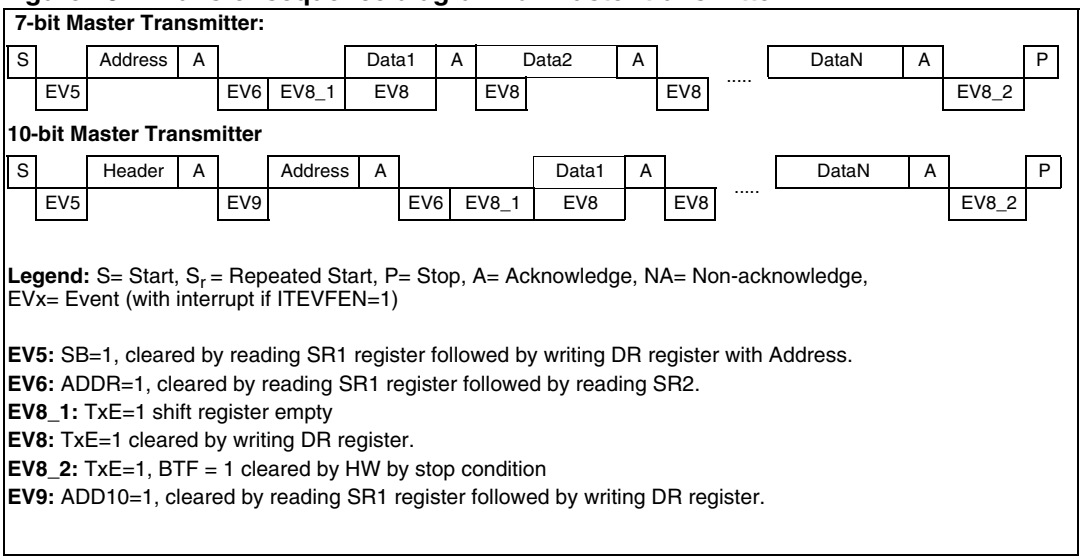
If TxE is set and a data byte was not written in the DR register before the end of the last data transmission, BTF is set and the interface waits until BTF is cleared.

Closing the communication

After writing the last byte to the DR register, the STOP bit is set by software to generate a Stop condition (see *Figure 234* Transfer sequencing EV8_2). The interface goes automatically back to slave mode (M/SL bit cleared).

Note: Stop condition should be programmed during EV8_2 event, when either TxE or BTF is set.

Figure 234. Transfer sequence diagram for master transmitter



Master receiver

Following the address transmission and after clearing ADDR, the I^2C interface enters Master Receiver mode. In this mode the interface receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RxNE bit is set and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set (see *Figure 235* Transfer sequencing EV7).

If the RxNE bit is set and the data in the DR register is not read before the end of the last data reception, the BTF bit is set by hardware and the interface waits for a read in the DR register.

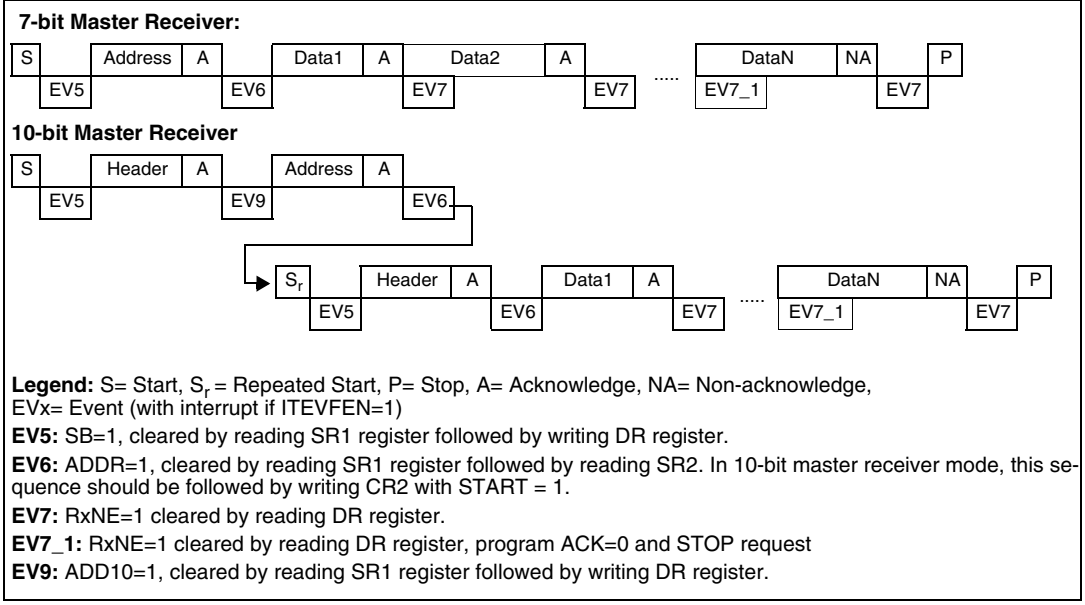
Closing the communication

The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a Stop/Re-Start condition.

- In order to generate the non-acknowledge pulse after the last received data byte, the ACK bit must be cleared just after reading the second last data byte (after second last RxNE event).
- In order to generate the Stop/Re-Start condition, software must set the STOP/START bit just after reading the second last data byte (after the second last RxNE event).
- In case a single byte is to be received, the Acknowledge disable and the Stop condition generation are made in EV6.

After the Stop condition generation, the interface goes automatically back to slave mode (M/SL bit cleared).

Figure 235. Transfer sequence diagram for master receiver



23.3.4 Error conditions

The following are the error conditions which may cause communication to fail.

Bus error (BERR)

This error occurs when the I^2C interface detects a Stop or a Start condition during a byte transfer. In this case,

- The BERR bit is set and an interrupt is generated if the ITERREN bit is set
- In case of Slave: data is discarded and the lines are released by hardware:
  - in case of misplaced start, the slave considers it is a restart and waits for address, or stop condition.
  - in case of misplaced stop, the slave reacts like for a stop condition and the lines are released by hardware.

Acknowledge failure (AF)

This error occurs when the interface detects a non-acknowledge bit. In this case,

- The AF bit is set and an interrupt is generated if the ITERREN bit is set
- A transmitter which receives a NACK must reset the communication:
  - If Slave: lines are released by hardware
  - If Master: a Stop condition must be generated by software

Arbitration lost (ARLO)

This error occurs when the I^2C interface detects an arbitration lost condition. In this case,

- The ARLO bit is set by hardware (and an interrupt is generated if the ITERREN bit is set)
- The I^2C Interface goes automatically back to slave mode (the M/SL bit is cleared)
- Lines are released by hardware

Overrun/underrun error (OVR)

An Overrun error can occur in slave mode when clock stretching is disabled and the I^2C interface is receiving data. The interface has received a byte (RxNE=1) and the data in DR has not been read, before the next byte is received by the interface. In this case,

- The last received byte is lost.
- In case of Overrun error, software should clear the RxNE bit and the transmitter should re-transmit the last received byte.

Underrun error can occur in slave mode when clock stretching is disabled and the I^2C interface is transmitting data. The interface has not updated the DR with the next byte (TxE=1), before the clock comes for the next byte. In this case,

- The same byte in the DR register will be sent again
- The user should make sure that data received on the receiver side during an underrun error is discarded and that the next bytes are written within the clock low time specified in the I^2C bus standard.

23.3.5 SDA/SCL line control

- If clock stretching is enabled:
  - Transmitter mode: If TxE=1 and BTF=1: the interface holds the clock line low before transmission to wait for the microcontroller to read SR1 and then write the byte in the Data Register (both buffer and shift register are empty).
  - Receiver mode: If RxNE=1 and BTF=1: the interface holds the clock line low after reception to wait for the microcontroller to read SR1 and then read the byte in the Data Register (both buffer and shift register are full).
- If clock stretching is disabled in Slave mode:
  - Overrun Error in case of RxNE=1 and no read of DR has been done before the next byte is received. The last received byte is lost.
  - Underrun Error in case TxE=1 and no write into DR has been done before the next byte must be transmitted. The same byte will be sent again.
  - Write Collision not managed.

23.3.6 SMBus

Introduction

The System Management Bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on I^2C principles of operation. SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass messages to and from devices instead of toggling individual control lines.

The System Management Bus Specification refers to three types of devices. A *slave* is a device that is receiving or responding to a command. A *master* is a device that issues commands, generates the clocks, and terminates the transfer. A *host* is a specialized master that provides the main interface to the system's CPU. A host must be a master-slave and must support the SMBus host notify protocol. Only one host is allowed in a system.

Similarities between SMBus and I^2C

- 2 wire bus protocol (1 Clk, 1 Data) + SMBus Alert line optional
- Master-slave communication, Master provides clock
- Multi master capability
- SMBus data format similar to I^2C 7-bit addressing format (*Figure 230*).

Differences between SMBus and I^2C

The following table describes the differences between SMBus and I^2C.

Table 150. SMBus vs. I^2C

SMBus	I^2C
Max. speed 100 kHz	Max. speed 400 kHz
Min. clock speed 10 kHz	No minimum clock speed
35 ms clock low timeout	No timeout
Logic levels are fixed	Logic levels are VDD dependent
Different address types (reserved, dynamic etc.)	7-bit, 10-bit and general call slave address types
Different bus protocols (quick command, process call etc.)	No bus protocols

SMBus application usage

With System Management Bus, a device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status. SMBus provides a control bus for system and power management related tasks.

Device identification

Any device that exists on the System Management Bus as a slave has a unique address called the Slave Address. For the list of reserved slave addresses, refer to the SMBus specification ver. 2.0 (*http://smbus.org/specs/*).

Bus protocols

The SMBus specification supports up to 9 bus protocols. For more details of these protocols and SMBus address types, refer to SMBus specification ver. 2.0 (*http://smbus.org/specs/*). These protocols should be implemented by the user software.

Address resolution protocol (ARP)

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. The Address Resolution Protocol (ARP) has the following attributes:

- Address assignment uses the standard SMBus physical layer arbitration mechanism
- Assigned addresses remain constant while device power is applied; address retention through device power loss is also allowed
- No additional SMBus packet overhead is incurred after address assignment. (i.e. subsequent accesses to assigned slave addresses have the same overhead as accesses to fixed address devices.)
- Any SMBus master can enumerate the bus

Unique device identifier (UDID)

In order to provide a mechanism to isolate each device for the purpose of address assignment, each device must implement a unique device identifier (UDID).

For the details on 128 bit UDID and more information on ARP, refer to SMBus specification ver. 2.0 (*http://smbus.org/specs/*).

SMBus alert mode

SMBus Alert is an optional signal with an interrupt line for devices that want to trade their ability to master for a pin. SMBALERT is a wired-AND signal just as the SCL and SDA signals are. SMBALERT is used in conjunction with the SMBus General Call Address. Messages invoked with the SMBus are 2 bytes long.

A slave-only device can signal the host through SMBALERT that it wants to talk by setting ALERT bit in I2C_CR1 register. The host processes the interrupt and simultaneously accesses all SMBALERT devices through the *Alert Response Address* (known as ARA having a value 0001 100X). Only the device(s) which pulled SMBALERT low will acknowledge the Alert Response Address. This status is identified using SMBALERT Status flag in I2C_SR1 register. The host performs a modified Receive Byte operation. The 7 bit device address provided by the slave transmit device is placed in the 7 most significant bits of the byte. The eighth bit can be a zero or one.

If more than one device pulls SMBALERT low, the highest priority (lowest address) device will win communication rights via standard arbitration during the slave address transfer. After acknowledging the slave address the device must disengage its SMBALERT pull-down. If the host still sees SMBALERT low when the message transfer is complete, it knows to read the ARA again.
A host which does not implement the SMBALERT signal may periodically access the ARA.

For more details on SMBus Alert mode, refer to SMBus specification ver. 2.0 (*http://smbus.org/specs/*).

Timeout error

There are differences in the timing specifications between I^2C and SMBus.
SMBus defines a clock low timeout, TIMEOUT of 35 ms. Also SMBus specifies TLOW: SEXT as the cumulative clock low extend time for a slave device. SMBus specifies TLOW: MEXT as the cumulative clock low extend time for a master device. For more details on these timeouts, refer to SMBus specification ver. 2.0 (*http://smbus.org/specs/*).

The status flag Timeout or Tlow Error in I2C_SR1 shows the status of this feature.

How to use the interface in SMBus mode

To switch from I^2C mode to SMBus mode, the following sequence should be performed.

- Set the SMBus bit in the I2C_CR1 register
- Configure the SMBTYPE and ENARP bits in the I2C_CR1 register as required for the application

If you want to configure the device as a master, follow the Start condition generation procedure in *Section 23.3.3: I2C master mode*. Otherwise, follow the sequence in *Section 23.3.2: I2C slave mode*.

The application has to control the various SMBus protocols by software.

- SMB Device Default Address acknowledged if ENARP=1 and SMBTYPE=0
- SMB Host Header acknowledged if ENARP=1 and SMBTYPE=1
- SMB Alert Response Address acknowledged if SMBALERT=1

23.3.7 DMA requests

DMA requests (when enabled) are generated only for data transfer. DMA requests are generated by Data Register becoming empty in transmission and Data Register becoming full in reception. When the number of data transfers which has been programmed for the corresponding DMA channel is reached, the DMA controller sends an End of Transfer EOT signal to the I^2C interface and generates a Transfer Complete interrupt if enabled:

- Master Transmitter: In the interrupt routine after the EOT interrupt, disable DMA requests then wait for a BTF event before programming the Stop condition.
- Master Receiver: The DMA controller sends a hardware signal EOT_1 corresponding to the (number of bytes -1). If, in the I2C_CR2 register, the LAST bit is set, I^2C automatically sends a NACK after the next byte following EOT_1. The user can generate a Stop condition in the DMA Transfer Complete interrupt routine if enabled.

Note: Please refer to the product specs for availability DMA controller. If DMA is not available in the product, the user should use I^2C as explained in section 1.4. In the I^2C ISR, the user can clear TxE/ RxNE flags to achieve continuous communication.

Transmission using DMA

DMA mode can be enabled for transmission by setting the DMAEN bit in the I2C_CR2 register. Data will be loaded from a Memory area configured using the DMA peripheral (refer to the DMA specification) to the I2C_DR register whenever the TxE bit is set. To map a DMA channel for I^2C transmission, perform the following sequence. Here x is the channel number.

1. Set the I2C_DR register address in the DMA_CPARx register. The data will be moved to this address from the memory after each TxE event.
2. Set the memory address in the DMA_CMARx register. The data will be loaded into I2C_DR from this memory after each TxE event.
3. Configure the total number of bytes to be transferred in the DMA_CNDTRx register. After each TxE event, this value will be decremented.
4. Configure the channel priority using the PL[0:1] bits in the DMA_CCRx register
5. Set the DIR bit and, in the DMA_CCRx register, configure interrupts after half transfer or full transfer depending on application requirements.
6. Activate the channel by setting the EN bit in the DMA_CCRx register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT_1 signal to the I^2C interface and the DMA generates an interrupt, if enabled, on the DMA channel interrupt vector.

Note: Do not enable the ITBUFEN bit in the I2C_CR2 register if DMA is used for transmission.

Reception using DMA

DMA mode can be enabled for reception by setting the DMAEN bit in the I2C_CR2 register. Data will be loaded from the I2C_DR register to a Memory area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA channel for I^2C reception, perform the following sequence. Here x is the channel number.

1. Set the I2C_DR register address in DMA_CPARx register. The data will be moved from this address to the memory after each RxNE event.
2. Set the memory address in the DMA_CMARx register. The data will be loaded from the I2C_DR register to this memory area after each RxNE event.
3. Configure the total number of bytes to be transferred in the DMA_CNDTRx register. After each RxNE event, this value will be decremented.
4. Configure the channel priority using the PL[0:1] bits in the DMA_CCRx register
5. Reset the DIR bit and configure interrupts in the DMA_CCRx register after half transfer or full transfer depending on application requirements.
6. Activate the channel by setting the EN bit in the DMA_CCRx register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT_1 signal to the I^2C interface and DMA generates an interrupt, if enabled, on the DMA channel interrupt vector.

Note: Do not enable the ITBUFEN bit in the I2C_CR2 register if DMA is used for reception.

23.3.8 Packet error checking

A PEC calculator has been implemented to improve the reliability of communication. The PEC is calculated by using the $C(x) = x^8 + x^2 + x + 1$ CRC-8 polynomial serially on each bit.

- PEC calculation is enabled by setting the ENPEC bit in the I2C_CR1 register. PEC is a CRC-8 calculated on all message bytes including addresses and R/W bits.
  - In transmission: set the PEC transfer bit in the I2C_CR1 register after the TxE event corresponding to the last byte. The PEC will be transferred after the last transmitted byte.
  - In reception: set the PEC bit in the I2C_CR1 register after the RxNE event corresponding to the last byte so that the receiver sends a NACK if the next received byte is not equal to the internally calculated PEC. In case of Master-Receiver, a NACK must follow the PEC whatever the check result.
- A PECERR error flag/interrupt is also available in the I2C_SR1 register.
- If DMA and PEC calculation are both enabled:-
  - In transmission: when the I^2C interface receives an EOT signal from the DMA controller, it automatically sends a PEC after the last byte.
  - In reception: when the I^2C interface receives an EOT_1 signal from the DMA controller, it will automatically consider the next byte as a PEC and will check it. A DMA request is generated after PEC reception.

- To allow intermediate PEC transfers, a control bit is available in the I2C_CR2 register (LAST bit) to determine if it is really the last DMA transfer or not. If it is the last DMA request for a master receiver, a NACK is automatically sent after the last received byte.
- PEC calculation is corrupted by an arbitration loss.

23.4 I^2C interrupts

The table below gives the list of I^2C interrupt requests.

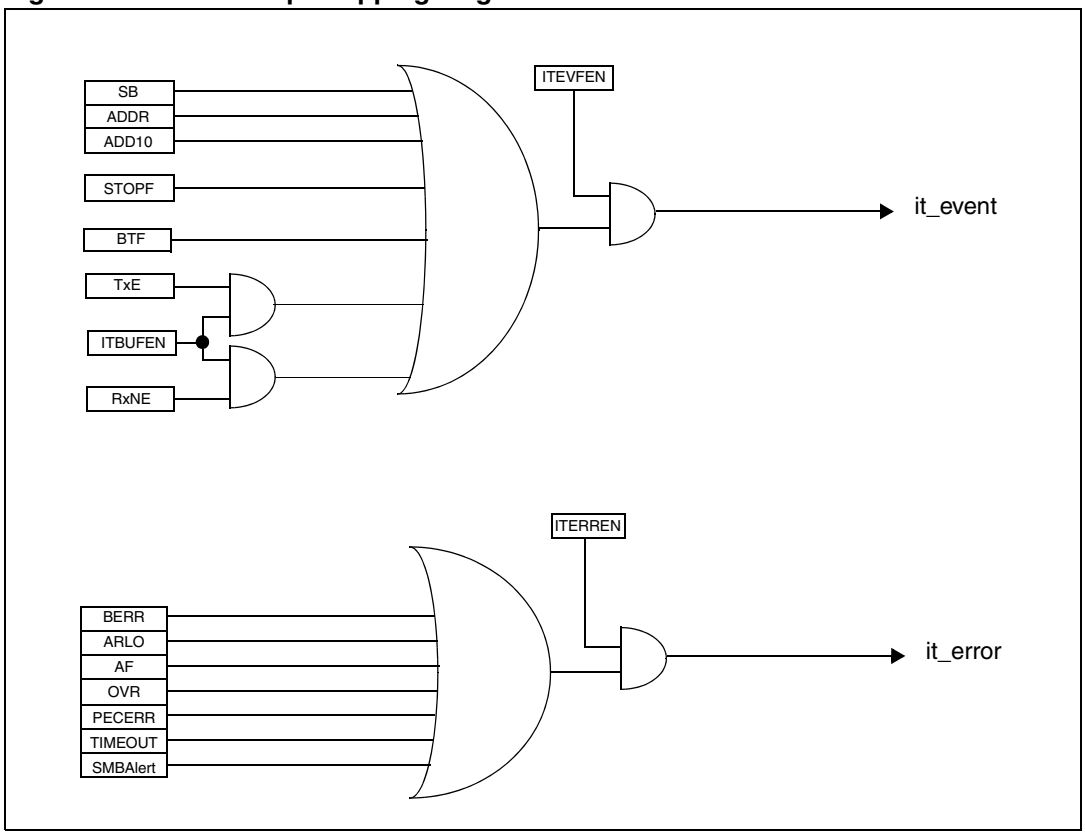
Table 151. I^2C Interrupt requests

Interrupt event	Event flag	Enable Control bit
Start bit sent (Master)	SB	ITEVFEN
Address sent (Master) or Address matched (Slave)	ADDR	
10-bit header sent (Master)	ADD10	
Stop received (Slave)	STOPF	
Data Byte Transfer Finished	BTF	
Receive buffer not empty	RxNE	ITEVFEN and ITBUFEN
Transmit buffer empty	TxE	
Bus error	BERR	ITERREN
Arbitration loss (Master)	ARLO	
Acknowledge failure	AF	
Overrun/Underrun	OVR	
PEC error	PECERR	
Timeout/Tlow error	TIMEOUT	
SMBus Alert	SMBALERT	

Note: 1 SB, ADDR, ADD10, STOPF, BTF, RxNE and TxE are logically ORed on the same interrupt channel.

2 BERR, ARLO, AF, OVR, PECERR, TIMEOUT and SMBALERT are logically ORed on the same interrupt channel.

Figure 236. I²C interrupt mapping diagram



23.5 I²C debug mode

When the microcontroller enters the debug mode (Cortex-M3 core halted), the SMBUS timeout either continues to work normally or stops, depending on the DBG_I2Cx_SMBUS_TIMEOUT configuration bits in the DBG module. For more details, refer to *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C on page 658*.

23.6 I²C registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

23.6.1 Control register 1(I2C_CR1)

Address offset: 0x00
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW RST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	EN PEC	EN ARP	SMB TYPE	Res.	SM BUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bit 15 **SWRST**: *Software Reset*

When set, the I2C is under reset state. Before resetting this bit, make sure the I2C lines are released and the bus is free.
0: I^2C Peripheral not under reset
1: I^2C Peripheral under reset state

Note:
This bit can be used in case the BUSY bit is set to '1' when no stop condition has been detected on the bus.

Bit 14 Reserved, forced by hardware to 0.

Bit 13 **ALERT**: *SMBus Alert*

This bit is set and cleared by software, and cleared by hardware when PE=0.
0: Releases SMBAlert pin high. Alert Response Address Header followed by NACK.
1: Drives SMBAlert pin low. Alert Response Address Header followed by ACK.

Bit 12 **PEC**: *Packet Error Checking.*

This bit is set and cleared by software, and cleared by hardware when PEC is transferred or by a START or Stop condition or when PE=0.
0: No PEC transfer
1: PEC transfer (in Tx or Rx mode)

Note: PEC calculation is corrupted by an arbitration loss.

Bit 11 **POS**: *Acknowledge/PEC Position (for data reception).*

This bit is set and cleared by software and cleared by hardware when PE=0.

0: ACK bit controls the (N)ACK of the current byte being received in the shift register. The PEC bit indicates that current byte in shift register is a PEC.
1: ACK bit controls the (N)ACK of the next byte which will be received in the shift register. The PEC bit indicates that the next byte in the shift register is a PEC

Note:
This bit must be configured before data reception starts.
This configuration must be used only in ADDR stretch event in case there are only 2 data bytes

Bit 10 **ACK**: *Acknowledge Enable*

This bit is set and cleared by software and cleared by hardware when PE=0.
0: No acknowledge returned
1: Acknowledge returned after a byte is received (matched address or data)

Bit 9 **STOP**: *Stop Generation*

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.
In Master Mode:
0: No Stop generation.
1: Stop generation after the current byte transfer or after the current Start condition is sent.
In Slave mode:
0: No Stop generation.
1: Release the SCL and SDA lines after the current byte transfer.

Note:
In Master mode, the BTF bit of the I2C_SR1 register must be cleared when Stop is requested.

Bit 8 **START**: *Start Generation*

This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.
In Master Mode:
0: No Start generation
1: Repeated start generation
In Slave mode:
0: No Start generation
1: Start generation when the bus is free

Bit 7 **NOSTRETCH**: *Clock Stretching Disable (Slave mode)*

This bit is used to disable clock stretching in slave mode when ADDR or BTF flag is set, until it is reset by software.
0: Clock stretching enabled
1: Clock stretching disabled

Bit 6 **ENGC**: *General Call Enable*

0: General call disabled. Address 00h is NACKed.
1: General call enabled. Address 00h is ACKed.

Bit 5 **ENPEC**: *PEC Enable*

0: PEC calculation disabled
1: PEC calculation enabled

Bit 4 **ENARP**: *ARP Enable*

0: ARP disable
1: ARP enable
SMBus Device default address recognized if SMBTYPE=0
SMBus Host address recognized if SMBTYPE=1

Bit 3 **SMBTYPE**: *SMBus Type*

0: SMBus Device
1: SMBus Host

Bit 2 Reserved, forced by hardware to 0.

Bit 1 **SMBUS**: *SMBus Mode*

0: I^2C mode
1: SMBus mode

Bit 0 **PE**: *Peripheral Enable*

0: Peripheral disable
1: Peripheral enable: the corresponding I/Os are selected as alternate functions depending on SMBus bit.

Note:

If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.
All bit resets due to PE=0 occur at the end of the communication.

In master mode, this bit must not be reset before the end of the communication.

23.6.2 Control register 2 (I2C_CR2)

Address offset: 0x04
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			LAST	DMA EN	ITBUF EN	ITEVT EN	ITER REN	Reserved		FREQ[5:0]					
Res.			rw	rw	rw	rw	rw	Res.		rw	rw	rw	rw	rw	rw

Bits 15:13 Reserved, forced by hardware to 0.

Bit 12 **LAST**: *DMA Last Transfer*

0: Next DMA EOT is not the last transfer
1: Next DMA EOT is the last transfer

Note: This bit is used in master receiver mode to permit the generation of a NACK on the last received data.

Bit 11 **DMAEN**: *DMA Requests Enable*

0: DMA requests disabled
1: DMA request enabled when TxE=1 or RxNE =1

Bit 10 **ITBUFEN**: *Buffer Interrupt Enable*

0: TxE = 1 or RxNE = 1 does not generate any interrupt.
1:TxE = 1 or RxNE = 1 generates Event Interrupt (whatever the state of DMAEN)

Bit 9 **ITEVTEN**: *Event Interrupt Enable*

0: Event interrupt disabled
1: Event interrupt enabled

This interrupt is generated when:

- SB = 1 (Master)
- ADDR = 1 (Master/Slave)
- ADD10= 1 (Master)
- STOPF = 1 (Slave)
- BTF = 1 with no TxE or RxNE event
- TxE event to 1 if ITBUFEN = 1
- RxNE event to 1if ITBUFEN = 1

Bit 8 **ITERREN**: *Error Interrupt Enable*

0: Error interrupt disabled
1: Error interrupt enabled

This interrupt is generated when:

- BERR = 1
- ARLO = 1
- AF = 1
- OVR = 1
- PECERR = 1
- TIMEOUT = 1
- SMBAlert = 1

Bits 7:6 Reserved, forced by hardware to 0.

Bits 5:0 **FREQ[5:0]**: *Peripheral Clock Frequency*

Input clock frequency must be programmed to generate correct timings
The allowed range is between 2 MHz and 36 MHz
000000: Not allowed
000001: Not allowed
000010: 2 MHz
...
100100: 36 MHz
Higher than 100100: Not allowed

23.6.3 Own address register 1 (I2C_OAR1)

Reset Address offset: 0x08
Value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD MODE	Res.	Reserved				ADD[9:8]		ADD[7:1]							ADD0
rw		Res.				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **ADDMODE** *Addressing Mode (Slave mode)*

0: 7-bit slave address (10-bit address not acknowledged)
1: 10-bit slave address (7-bit address not acknowledged)

Bit 14 Must be configured and kept at 1.

Bits 13:10 Reserved, forced by hardware to 0.

Bits 9:8 **ADD[9:8]**: *Interface Address*

7-bit addressing mode: don't care
10-bit addressing mode: bits9:8 of address

Bits 7:1 **ADD[7:1]**: *Interface Address*

bits 7:1 of address

Bit 0 **ADD0**: *Interface Address*

7-bit addressing mode: don't care
10-bit addressing mode: bit 0 of address

23.6.4 Own address register 2 (I2C_OAR2)

Address offset: 0x0C
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								ADD2[7:1]							ENDUAL
Res.								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved, forced by hardware to 0.

Bits 7:1 **ADD2[7:1]**: *Interface address*
bits 7:1 of address in dual addressing mode

Bit 0 **ENDUAL**: *Dual addressing mode enable*
0: Only OAR1 is recognized in 7-bit addressing mode
1: Both OAR1 and OAR2 are recognized in 7-bit addressing mode

23.6.5 Data register (I2C_DR)

Address offset: 0x10
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[7:0]							
Res.								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved, forced by hardware to 0.

Bits 7:0 **DR[7:0]** 8-bit *Data Register* (1)(2)(3)
Byte received or to be transmitted to the bus.
– Transmitter mode: Byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in DR once the transmission is started (TxE=1)
– Receiver mode: Received byte is copied into DR (RxNE=1). The received data in the DR register must be read before the next data reception, otherwise an overrun occurs and the last byte will be lost.

1. In slave mode, the address is not copied into DR.
2. Write collision is not managed (DR can be written if TxE=0).
3. If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.

23.6.6 Status register 1 (I2C_SR1)

Address offset: 0x14
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOP F	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 15 **SMBALERT**: *SMBus Alert*

In SMBus host mode:
0: no SMBAlert
1: SMBAlert event occurred on pin
In SMBus slave mode:
0: no SMBAlert response address header
1: SMBAlert response address header to SMBAlert LOW received

– Cleared by software writing 0, or by hardware when PE=0.

Bit 14 **TIMEOUT**: *Timeout or Tlow Error*

0: No timeout error
1: SCL remained LOW for 25 ms (Timeout)
or
Master cumulative clock low extend time more than 10 ms (Tlow:mext)
or
Slave cumulative clock low extend time more than 25 ms (Tlow:sext)

– When set in slave mode: slave resets the communication and lines are released by hardware
– When set in master mode: Stop condition sent by hardware
– Cleared by software writing 0, or by hardware when PE=0.

Bit 13 Reserved, forced by hardware to 0.

Bit 12 **PECERR**: *PEC Error in reception*

0: no PEC error: receiver returns ACK after PEC reception (if ACK=1)
1: PEC error: receiver returns NACK after PEC reception (whatever ACK)
– Cleared by software writing 0, or by hardware when PE=0.

Bit 11 **OVR**: *Overrun/Underrun*

0: No overrun/underrun
1: Overrun or underrun
– Set by hardware in slave mode when NOSTRETCH=1 and:
– In reception when a new byte is received (including ACK pulse) and the DR register has not been read yet. New received byte is lost.
– In transmission when a new byte should be sent and the DR register has not been written yet. The same byte is sent twice.
– Cleared by software writing 0, or by hardware when PE=0.

Note:
If the DR write occurs very close to SCL rising edge, the sent data is unspecified and a hold timing error occurs

Bit 10 **AF**: *Acknowledge Failure.*

0: No acknowledge failure
1: Acknowledge failure

– Set by hardware when no acknowledge is returned.
– Cleared by software writing 0, or by hardware when PE=0.

Bit 9 **ARLO**: *Arbitration Lost (master mode)*

0: No Arbitration Lost detected
1: Arbitration Lost detected

Set by hardware when the interface loses the arbitration of the bus to another master

– Cleared by software writing 0, or by hardware when PE=0.

After an ARLO event the interface switches back automatically to Slave mode (M/SL=0).

Note:
In SMBUS, the arbitration on the data in slave mode occurs only during the data phase, or the acknowledge transmission (not on the address acknowledge).

Bit 8 **BERR**: *Bus Error*

0: No misplaced Start or Stop condition
1: Misplaced Start or Stop condition

– Set by hardware when the interface detects a misplaced Start or Stop condition
– Cleared by software writing 0, or by hardware when PE=0.

Bit 7 **TxE**: *Data Register Empty (transmitters)*

0: Data register not empty
1: Data register empty

– Set when DR is empty in transmission. TxE is not set during address phase.
– Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

TxE is not set if either a NACK is received, or if next byte to be transmitted is PEC (PEC=1)

Bit 6 **RxNE**: *Data Register not Empty (receivers).*

0: Data register empty
1: Data register not empty

– Set when data register is not empty in receiver mode. RxNE is not set during address phase.
– Cleared by software reading or writing the DR register or by hardware when PE=0.

RxNE is not set in case of ARLO event.

Bit 5 Reserved, forced by hardware to 0.

Bit 4 **STOPF**: *Stop detection (Slave mode)*

0: No Stop condition detected
1: Stop condition detected

– Set by hardware when a Stop condition is detected on the bus by the slave after an acknowledge (if ACK=1).
– Cleared by software reading the SR1 register followed by a write in the CR1 register, or by hardware when PE=0

Note: The STOPF bit is not set after a NACK reception

Bit 3 **ADD10**: *10-bit header sent (Master mode)*

0: No ADD10 event occurred.
1: Master has sent first address byte (header).
– Set by hardware when the master has sent the first byte in 10-bit address mode.
– Cleared by software reading the SR1 register followed by a write in the DR register of the second address byte, or by hardware when PE=0.

Note: ADD10 bit is not set after a NACK reception

Bit 2 **BTF**: *Byte Transfer Finished.*

0: Data Byte transfer not done
1: Data Byte transfer succeeded
– Set by hardware when NOSTRETCH=0 and:
  – In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RxNE=1).
  – In transmission when a new byte should be sent and DR has not been written yet (TxE=1).
  – Cleared by software reading SR1 followed by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE=0.

Note: The BTF bit is not set after a NACK reception

The BTF bit is not set if next byte to be transmitted is the PEC (TRA=1 in I2C_SR2 register and PEC=1 in I2C_CR1 register)

Bit 1 **ADDR**: *Address sent (master mode)/matched (slave mode)*

This bit is cleared by software reading SR1 register followed reading SR2, or by hardware when PE=0.

Address Matched (Slave)

0: Address mismatched or not received.
1: Received address matched.
– Set by hardware as soon as the received slave address matched with the OAR registers content or a general call or a SMBus Device Default Address or SMBus Host or SMBus Alert is recognized. (when enabled depending on configuration).

Address Sent (Master)

0: No end of address transmission
1: End of address transmission
– For 10-bit addressing, the bit is set after the ACK of the 2nd byte.
– For 7-bit addressing, the bit is set after the ACK of the byte.

Note: ADDR is not set after a NACK reception

Bit 0 **SB**: *Start Bit (Master mode).*

0: No Start condition
1: Start condition generated.
– Set when a Start condition generated.
– Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

23.6.7 Status register 2 (I2C_SR2)

Address offset: 0x18
Reset value:0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMB DEF AULT	GEN CALL	Res.	TRA	BUSY	MSL
r	r	r	r	r	r	r	r	r	r	r	r		r	r	r

Bits 15:8 **PEC[7:0]** *Packet Error Checking Register*

This register contains the internal PEC when ENPEC=1.

Bit 7 **DUALF**: *Dual Flag (Slave mode)*

0: Received address matched with OAR1
1: Received address matched with OAR2
– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 6 **SMBHOST**: *SMBus Host Header (Slave mode)*

0: No SMBus Host address
1: SMBus Host address received when SMBTYPE=1 and ENARP=1.
– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 5 **SMBDEFAULT**: *SMBus Device Default Address (Slave mode)*

0: No SMBus Device Default address
1: SMBus Device Default address received when ENARP=1
– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 4 **GENCALL**: *General Call Address (Slave mode)*

0: No General Call
1: General Call Address received when ENGC=1
– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 3 Reserved, forced by hardware to 0.

Bit 2 **TRA**: *Transmitter/Receiver*

0: Data bytes received
1: Data bytes transmitted

This bit is set depending on the R/W bit of the address byte, at the end of total address phase.

It is also cleared by hardware after detection of Stop condition (STOPF=1), repeated Start condition, loss of bus arbitration (ARLO=1), or when PE=0.

Bit 1 **BUSY**: *Bus busy*

0: No communication on the bus
1: Communication ongoing on the bus
– Set by hardware on detection of SDA or SCL low
– cleared by hardware on detection of a Stop condition.

It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE=0).

Bit 0 **MSL**: *Master/slave*

0: Slave Mode
1: Master Mode
– Set by hardware as soon as the interface is in Master mode (SB=1).
– Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

23.6.8 Clock control register (I2C_CCR)

Address offset: 0x1C
Reset value: 0x0000

Note: 1 F_{PCLK1} is the multiple of 10 MHz required to generate the Fast clock at 400 kHz.
2 The CCR register must be configured only when the I2C is disabled (PE = 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **F/S** *I^2C Master Mode Selection*

0: Standard Mode I2C
1: Fast Mode I2C

Bit 14 **DUTY** *Fast Mode Duty Cycle*

0: Fast Mode t_{low}/t_{high} = 2
1: Fast Mode t_{low}/t_{high} = 16/9 (see CCR)

Bits 13:12 Reserved, forced by hardware to 0.

Bits 11:0 **CCR[11:0]** *Clock Control Register in Fast/Standard mode (Master mode)*

Controls the SCL clock in master mode.

Standard Mode or SMBus:

$T_{high} = CCR * T_{PCLK1}$

$T_{ow} = CCR * T_{PCLK1}$

Fast Mode:

If DUTY = 0:

$T_{high} = CCR * T_{PCLK1}$

$T_{ow} = 2 * CCR * T_{PCLK1}$

If DUTY = 1: (to reach 400 kHz)

$T_{high} = 9 * CCR * T_{PCLK1}$

$T_{ow} = 16 * CCR * T_{PCLK1}$

For instance: in standard mode, to generate a 100 kHz SCL frequency:

If FREQR = 08, T_{PCLK1} = 125 ns so CCR must be programmed with 0x28

(0x28 <=> 40d x 125 ns = 5000 ns.)

Note: 1. The minimum allowed value is 0x04, except in FAST DUTY mode where the minimum allowed value is 0x01

2. t_{high} includes the SCLH rising edge

3. t_{low} includes the SCLH falling edge

4. These timings are without filters.

5. The CCR register must be configured only when the I^2C is disabled (PE = 0).

6. f_{CK} = a multiple of 10 MHz is required to generate the fast clock at 400 kHz.

23.6.9 TRISE Register (I2C_TRISE)

Address offset: 0x20
Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										TRISE[5:0]					
Res.										rw	rw	rw	rw	rw	rw

Bits 15:6 Reserved, forced by hardware to 0.

Bits 5:0 **TRISE[5:0]**: *Maximum Rise Time in Fast/Standard mode (Master mode)*

These bits must be programmed with the maximum SCL rise time given in the I^2C bus specification, incremented by 1.

For instance: in standard mode, the maximum allowed SCL rise time is 1000 ns.

If, in the I2C_CR2 register, the value of FREQ[5:0] bits is equal to 0x08 and T_{PCLK1} = 125 ns therefore the TRISE[5:0] bits must be programmed with 09h.

(1000 ns / 125 ns = 8 + 1)

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Note: TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).

23.6.10 I²C register map

The table below provides the I²C register map and reset values.

Table 152. I²C register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	I2C_CR1	Reserved																SWRST	Reserved	ALERT	PEC	POS	ACK	STOP	START	NOSTRETCH	ENGC	ENPEC	ENARP	SMBTYPE	Reserved	SMBUS	PE
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0		0	0
0x04	I2C_CR2	Reserved																			LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]					
	Reset value																				0	0	0	0	0			0	0	0	0	0	0
0x08	I2C_OAR1	Reserved																ADDMODE	Reserved	Reserved				ADD[9:8]		ADD[7:1]							ADD0
	Reset value																	0	1					0	0	0	0	0	0	0	0	0	0
0x0C	I2C_OAR2	Reserved																								ADD2[7:1]							ENDUAL
	Reset value																									0	0	0	0	0	0	0	0
0x10	I2C_DR	Reserved																								DR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x14	I2C_SR1	Reserved																SMBALERT	TIMEOUT	Reserved	PECERR	OVR	AF	ARLO	BERR	TxE	RxNE	Reserved	STOPF	ADD10	BTF	ADDR	SB
	Reset value																	0	0		0	0	0	0	0	0	0		0	0	0	0	0
0x18	I2C_SR2	Reserved																PEC[7:0]								DUALF	SMBHOST	SMBDEFAULT	GENCALL	Reserved	TRA	BUSY	MSL
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
0x1C	I2C_CCR	Reserved																F/S	DUTY	Reserved				CCR[11:0]									
	Reset value																	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	I2C_TRISE	Reserved																										TRISE[5:0]					
	Reset value																											0	0	0	0	1	0

Refer to *Table 1 on page 35* for the register boundary addresses.

24 Universal synchronous asynchronous receiver transmitter (USART)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

24.1 USART introduction

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.

It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication.

High speed data communication is possible by using the DMA for multibuffer configuration.

24.2 USART main features

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- Fractional baud rate generator systems
  - A common programmable transmit and receive baud rates up to 4.5 MBits/s
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
  - 13-bit break generation and 10/11 bit break detection when USART is hardware configured for LIN
- Transmitter clock output for synchronous transmission
- IrDA SIR Encoder Decoder
  - Support for 3/16 bit duration for normal mode
- Smartcard Emulation Capability
  - The Smartcard interface supports the asynchronous protocol Smartcards as defined in ISO 7816-3 standards
  - 0.5, 1.5 Stop Bits for Smartcard operation
- Single wire Half Duplex Communication

- Configurable multibuffer communication using DMA (direct memory access)
  - Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for Transmitter and Receiver
- Transfer detection flags:
  - Receive buffer full
  - Transmit buffer empty
  - End of Transmission flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- Four error detection flags:
  - Overrun error
  - Noise error
  - Frame error
  - Parity error
- Ten interrupt sources with flags:
  - CTS changes
  - LIN break detection
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle line received
  - Overrun error
  - Framing error
  - Noise error
  - Parity error
- Multiprocessor communication - enter into mute mode if address match does not occur
- Wake up from mute mode (by idle line detection or address mark detection)
- Two receiver wakeup modes: Address bit (MSB, 9th bit), Idle line

24.3 USART functional description

The interface is externally connected to another device by three pins (see *Figure 237*). Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW_RX).

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data Register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART_GTPR) in case of Smartcard mode.

Refer to *Section 24.6: USART registers on page 625* for the definitions of each bit.

The following pin is required to interface in synchronous mode:

- **SCLK:** Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In smartcard mode, SCLK can provide the clock to the smartcard.

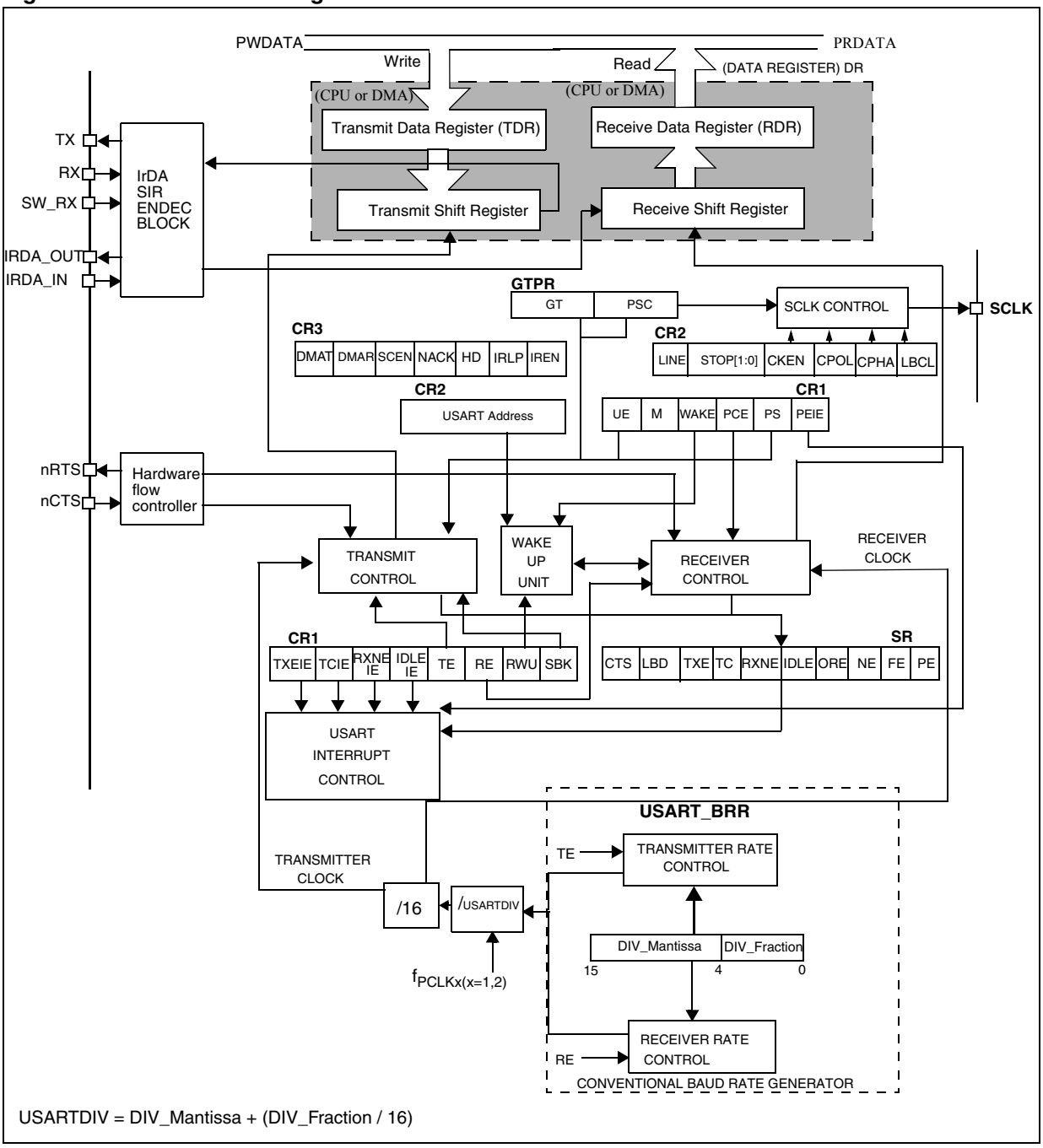
The following pins are required to interface in IrDA mode:

- **IrDA_RDI:** Receive Data Input is the data input in IrDA mode.
- **IrDA_TDO:** Transmit Data Output in IrDA mode.

the following pins are required in Hardware flow control mode:

- **nCTS:** Clear To Send blocks the data transmission at the end of the current transfer when high
- **nRTS:** Request to send indicates that the USART is ready to receive a data (when low).

Figure 237. USART block diagram



24.3.1 USART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the USART_CR1 register (see *Figure 238*).

The TX pin is in low state during the start bit. It is in high state during the stop bit.

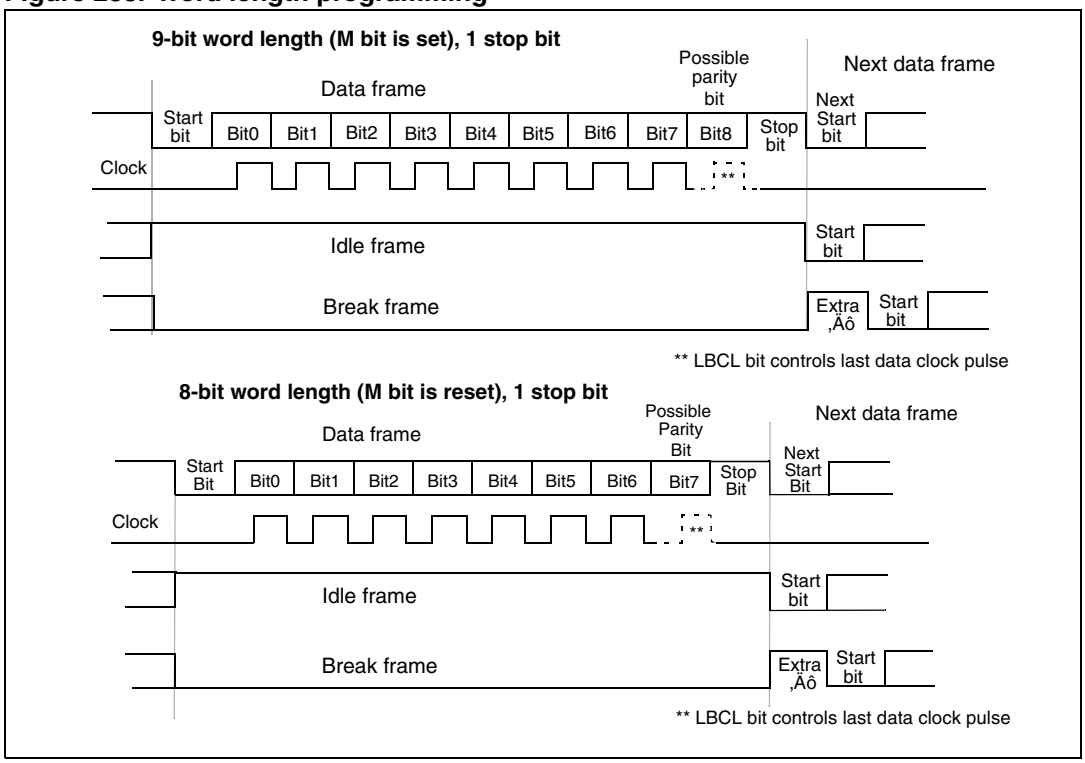
An ***Idle character*** is interpreted as an entire frame of “1”s followed by the start bit of the next frame which contains data (The number of “1” ‘s will include the number of stop bits).

A ***Break character*** is interpreted on receiving “0”s for a frame period. At the end of the break frame the transmitter inserts either 1 or 2 stop bits (logic “1” bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

Figure 238. Word length programming



24.3.2 Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the SCLK pin.

Character transmission

During an USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see *Figure 237*).

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART.

Note: 1 The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost.

2 An idle frame will be sent after the TE bit is enabled.

Configurable stop bits

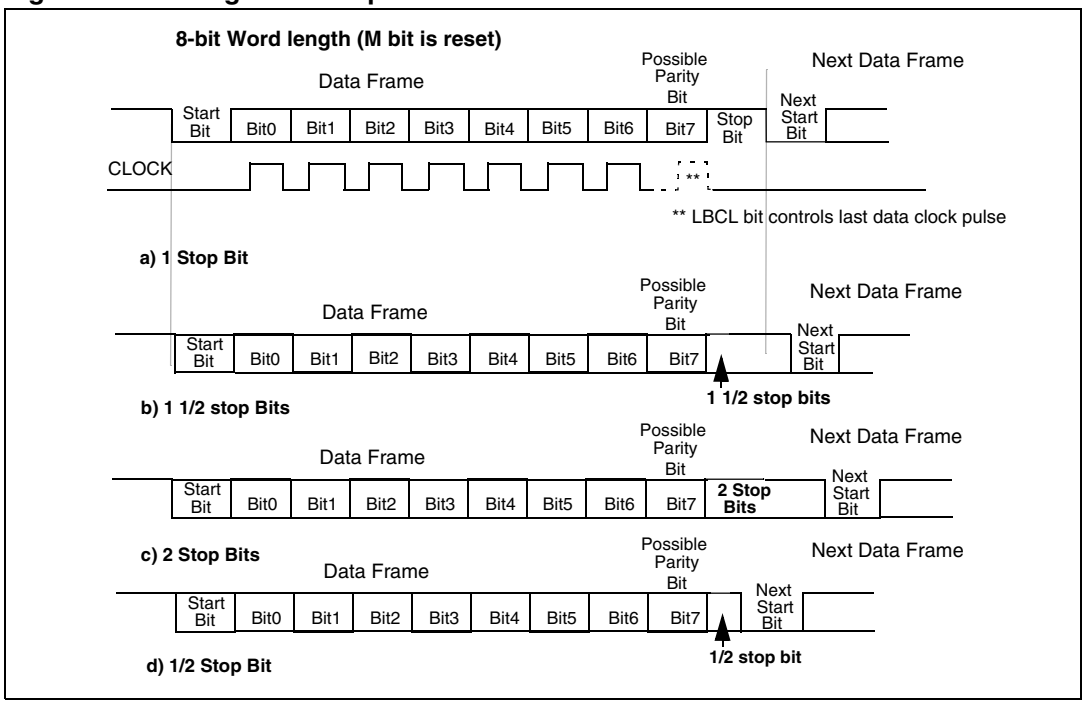
The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

1. ***1 stop bit***: This is the default value of number of stop bits.
2. ***2 Stop bits***: This will be supported by normal USART, single-wire and modem modes.
3. ***0.5 stop bit***: To be used when receiving data in Smartcard mode.
4. ***1.5 stop bits***: To be used when transmitting data in Smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission will be 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).

Figure 239. Configurable stop bits



Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
6. Select the desired baud rate using the USART_BRR register.
7. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.

Single byte communication

Clearing the TXE bit is always performed by a write to the data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

When a frame transmission is complete (after the stop bit) the TC bit is set and an interrupt is generated if the TCIE is set in the USART_CR1 register.

Clearing the TC bit is performed by the following software sequence:

1. A read to the USART_SR register
2. A write to the USART_DR register

Note: The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for Multibuffer communication.

Break characters

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see *Figure 238*).

If the SBK bit is set to '1' a break character is sent on the TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the stop bit of the break character). The USART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame.

Note: If the software resets the SBK bit before the commencement of break transmission, the break character will not be transmitted. For two consecutive breaks, the SBK bit should be set after the stop bit of the previous break.

Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

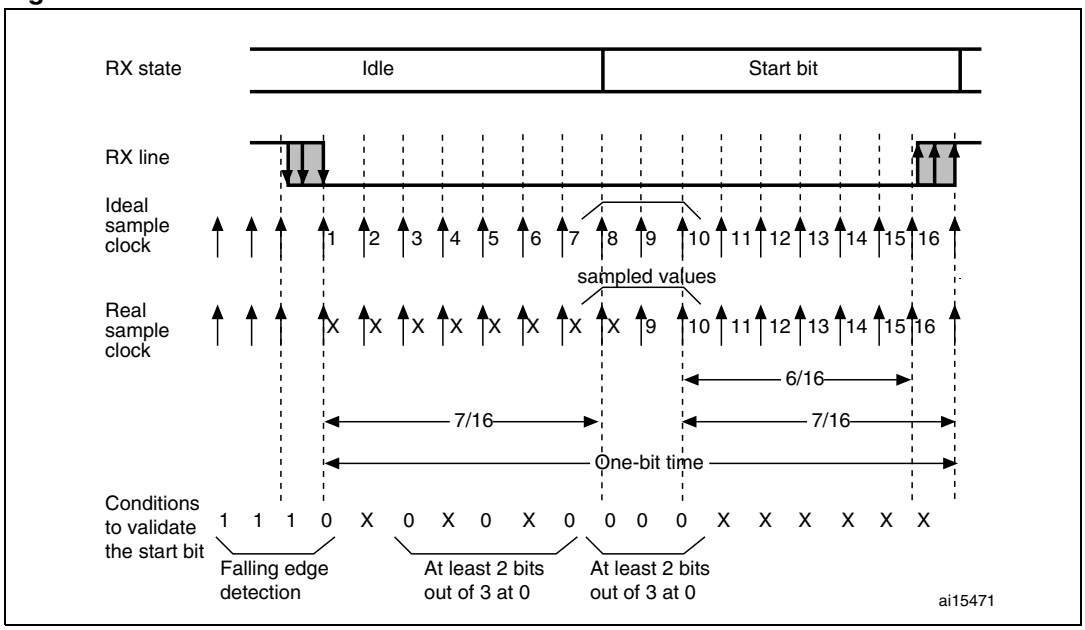
24.3.3 Receiver

The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Start bit detection

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0X 0X 0 X 0X 0.

Figure 240. Start bit detection



Note: If the sequence is not complete, the start bit detection aborts and the receiver returns to idle state (no flag is set) waiting for a falling edge.

If only 2 out of the 3 bits are at 0 (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), the start bit is validated but the NE noise flag bit is set.

The start bit is confirmed if the last 3 samples are at 0 (sampling on the 8th, 9th, and 10th bits.

Character reception

During an USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note: The RE bit should not be reset while receiving data. If the RE bit is disabled during reception, the reception of the current byte will be aborted.

Break character

When a break character is received, the USART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the IDLEIE bit is set.

Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced. When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_DR is performed.
- The shift register will be overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or both the EIE and DMAR bits are set.
- The ORE bit is reset by a read to the USART_SR register followed by a USART_DR register read operation.

Note: The ORE bit, when set, indicates that at least 1 data has been lost. There are two possibilities:

- *if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,*
- *if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received. It may also occur*

when the new data is received during the reading sequence (between the USART_SR register read access and the USART_DR read access).

Noise error

Over-sampling techniques are used (except in synchronous mode) for data recovery by discriminating between valid incoming data and noise.

Figure 241. Data sampling for noise detection

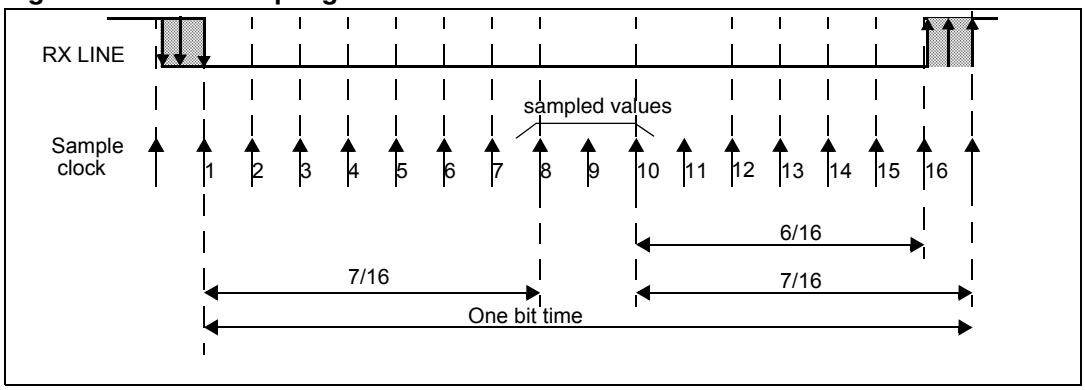


Table 153. Noise detection from sampled data

Sampled value	NE status	Received bit value	Data validity
000	0	0	Valid
001	1	0	Not Valid
010	1	0	Not Valid
011	1	1	Not Valid
100	1	0	Not Valid
101	1	1	Not Valid
110	1	1	Not Valid
111	0	1	Valid

When noise is detected in a frame:

- The NE is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART_DR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The NE bit is reset by a USART_SR register read operation followed by a USART_DR register read operation.

Framing error

A framing error is detected when:

The stop bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART_DR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The FE bit is reset by a USART_SR register read operation followed by a USART_DR register read operation.

Configurable stop bits during reception

The number of stop bits to be received can be configured through the control bits of Control Register 2 - it can be either 1 or 2 in normal mode and 0.5 or 1.5 in Smartcard mode.

1. ***0.5 stop Bit (reception in Smartcard mode)***: No sampling is done for 0.5 stop bit. As a consequence, no framing error and no break frame can be detected when 0.5 stop bit is selected.
2. ***1 stop Bit***: Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
3. ***1.5 stop Bits (transmission in Smartcard mode)***: When transmitting in smartcard mode, the device must check that the data is correctly sent. Thus the receiver block must be enabled (RE =1 in the USART_CR1 register) and the stop bit is checked to test if the smartcard has detected a parity error. In the event of a parity error, the smartcard forces the data signal low during the sampling - NACK signal-, which is flagged as a framing error. Then, the FE flag is set with the RXNE at the end of the 1.5 stop bit. Sampling for 1.5 stop bits is done on the 16th, 17th and 18th samples (1 baud clock period after the beginning of the stop bit). The 1.5 stop bit can be decomposed into 2 parts: one 0.5 baud clock period during which nothing happens, followed by 1 normal stop bit period during which sampling occurs halfway through. Refer to *Section 24.3.10: Smartcard on page 617* for more details.
4. ***2 stop Bits***: Sampling for 2 stop bits is done on the 8th, 9th and 10th samples of the first stop bit. If a framing error is detected during the first stop bit the framing error flag will be set. The second stop bit is not checked for framing error. The RXNE flag will be set at the end of the first stop bit.

24.3.4 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

legend: $f_{PCLKx(x=1,2)}$ - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

How to derive USARTDIV from USART_BRR register values

Example 1:

If DIV_Mantissa = 27d and DIV_Fraction= 12d (USART_BRR=1BCh), then

Mantissa (USARTDIV) = 27d

Fraction (USARTDIV) = 12/16 = 0.75d

Therefore USARTDIV = 27.75d

Example 2:

To program USARTDIV = 25.62d,

This leads to:

DIV_Fraction = 16*0.62d = 9.92d, nearest real number 10d = 0xA

DIV_Mantissa = mantissa (25.620d) = 25d = 0x19

Then, USART_BRR = 0x19A

Example 3:

To program USARTDIV = 50.99d

This leads to:

DIV_Fraction = 16*0.99d = 15.84d => nearest real number, 16d = 0x10

DIV_Mantissa = mantissa (50.990d) = 50d = 0x32

Note: The Baud Counters will be updated with the new value of the Baud Registers after a write to USART_BRR. Hence the Baud Rate Register value should not be changed during a transaction.

Table 154. Error calculation for programmed baud rates

Baud rate		f_{PCLK} = 36 MHz			f_{PCLK} = 72 MHz		
S.No	in Kbps	Actual	Value programmed in the Baud Rate register	% Error =(Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the Baud Rate register	% Error
1.	2.4	2.400	937.5	0%	2.4	1875	0%
2.	9.6	9.600	234.375	0%	9.6	468.75	0%
3.	19.2	19.2	117.1875	0%	19.2	234.375	0%
4.	57.6	57.6	39.0625	0%	57.6	78.125	0.%
5.	115.2	115.384	19.5	0.15%	115.2	39.0625	0%
6.	230.4	230.769	9.75	0.16%	230.769	19.5	0.16%
7.	460.8	461.538	4.875	0.16%	461.538	9.75	0.16%
8.	921.6	923.076	2.4375	0.16%	923.076	4.875	0.16%
9.	2250	2250	1	0%	2250	2	0%
10.	4500	NA	NA	NA	4500	1	0%

Note: 1 The lower the CPU clock the lower will be the accuracy for a particular Baud rate. The upper limit of the achievable baud rate can be fixed with this data.

2 Only USART1 is clocked with PCLK2 (72 MHz Max). Other USARTs are clocked with PCLK1 (36 MHz Max).

24.3.5 Multiprocessor communication

There is a possibility of performing multiprocessor communication with the USART (several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output is connected to the RX input of the other USART. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART_CR1 register is set to 1. RWU can be controlled automatically by hardware or written by the software under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART_CR1 register:

- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

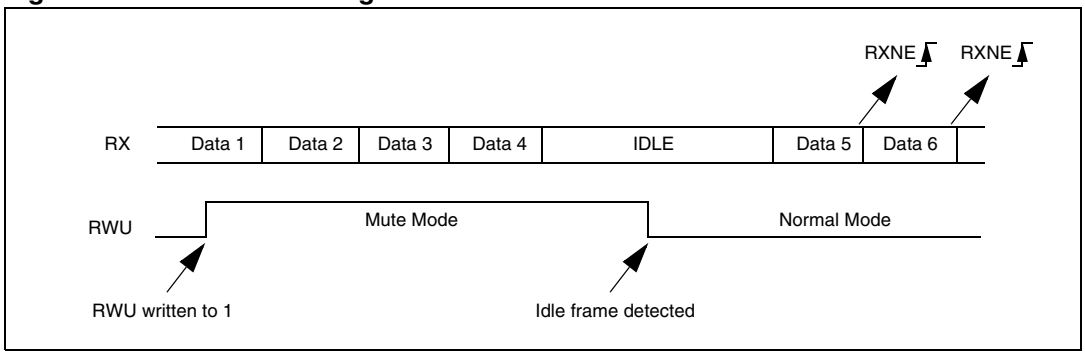
Idle line detection (WAKE=0)

The USART enters mute mode when the RWU bit is written to 1.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART_SR register. RWU can also be written to 0 by software.

An example of mute mode behavior using idle line detection is given in *Figure 242*.

Figure 242. Mute mode using Idle line detection



Address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a ‘1’ else they are considered as data. In an address byte, the address of the targeted receiver is put on the 4 LSB. This 4-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART_CR2 register.

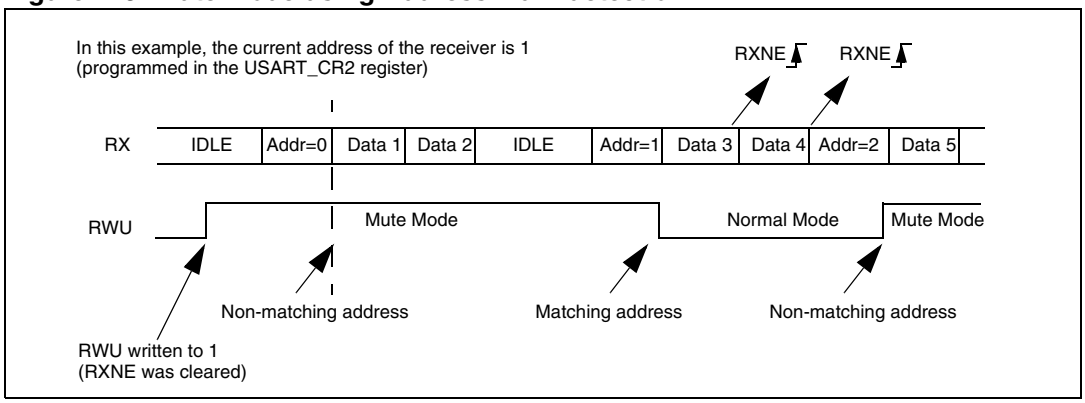
The USART enters mute mode when an address character is received which does not match its programmed address. The RXNE flag is not set for this address byte and no interrupt nor DMA request is issued as the USART would have entered mute mode.

It exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

The RWU bit can be written to as 0 or 1 when the receiver buffer contains no data (RXNE=0 in the USART_SR register). Otherwise the write attempt is ignored.

An example of mute mode behavior using address mark detection is given in *Figure 243*.

Figure 243. Mute mode using Address mark detection



24.3.6 Parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART_CR1 register. Depending on the frame length defined by the M bit, the possible USART frame formats are as listed in *Table 155*.

Table 155. Frame formats

M bit	PCE bit	USART frame
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7-bit data | PB | STB |
1	0	| SB | 9-bit data | STB |
1	1	| SB | 8-bit data PB | STB |

Legends: SB: Start Bit, STB: Stop Bit, PB: Parity Bit

Note: In case of wake up by an address mark, the MSB bit of the data is taken into account and not the parity bit

Even parity: the parity bit is calculated to obtain an even number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Ex: data=00110101; 4 bits set => parity bit will be 0 if even parity is selected (PS bit in USART_CR1 = 0).

Odd parity: the parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Ex: data=00110101; 4 bits set => parity bit will be 1 if odd parity is selected (PS bit in USART_CR1 = 1).

Transmission mode: If the PCE bit is set in USART_CR1, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit (even number of "1s" if even parity is selected (PS=0) or an odd number of "1s" if odd parity is selected (PS=1)). If the parity check fails, the PE flag is set in the USART_SR register and an interrupt is generated if PEIE is set in the USART_CR1 register.

24.3.7 LIN (local interconnection network) mode

The LIN mode is selected by setting the LINEN bit in the USART_CR2 register. In LIN mode, the following bits must be kept cleared:

- CLKEN in the USART_CR2 register,
- STOP[1:0], SCEN, HDSEL and IREN in the USART_CR3 register.

LIN transmission

The same procedure explained in *Section 24.3.2* has to be applied for LIN Master transmission than for normal USART transmission with the following differences:

- Clear the M bit to configure 8-bit word length.
- Set the LINEN bit to enter LIN mode. In this case, setting the SBK bit sends 13 '0' bits as a break character. Then a bit of value '1' is sent to allow the next start detection.

LIN reception

When the LIN mode is enabled, the break detection circuit is activated. The detection is totally independent from the normal USART receiver. A break can be detected whenever it occurs, during idle state or during a frame.

When the receiver is enabled (RE=1 in USART_CR1), the circuit looks at the RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 (when the LBDL = 0 in USART_CR2) or 11 (when LBDL=1 in USART_CR2) consecutive bits are detected as '0', and are followed by a delimiter character, the LBD flag is set in USART_SR. If the LBDIE bit=1, an interrupt is generated. Before validating the break, the delimiter is checked for as it signifies that the RX line has returned to a high level.

If a '1' is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again.

If the LIN mode is disabled (LINEN=0), the receiver continues working as normal USART, without taking into account the break detection.

If the LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at '0', which will be the case for any break frame), the receiver stops until the break detection circuit receives either a '1', if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown on the *Figure 244: Break detection in LIN mode (11-bit break length - LBDL bit is set) on page 613*.

Examples of break frames are given on *Figure 245: Break detection in LIN mode vs. Framing error detection on page 614*.

Figure 244. Break detection in LIN mode (11-bit break length - LBDL bit is set)

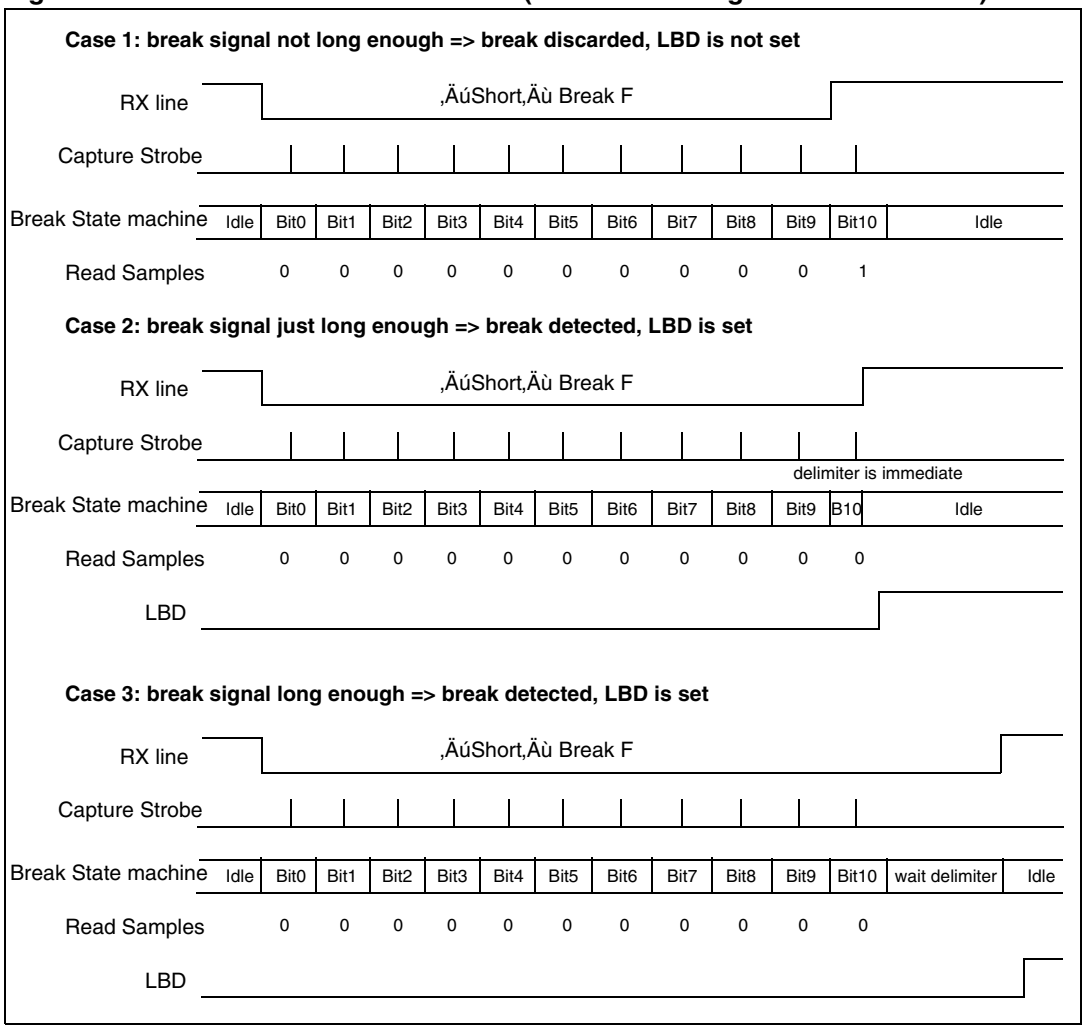
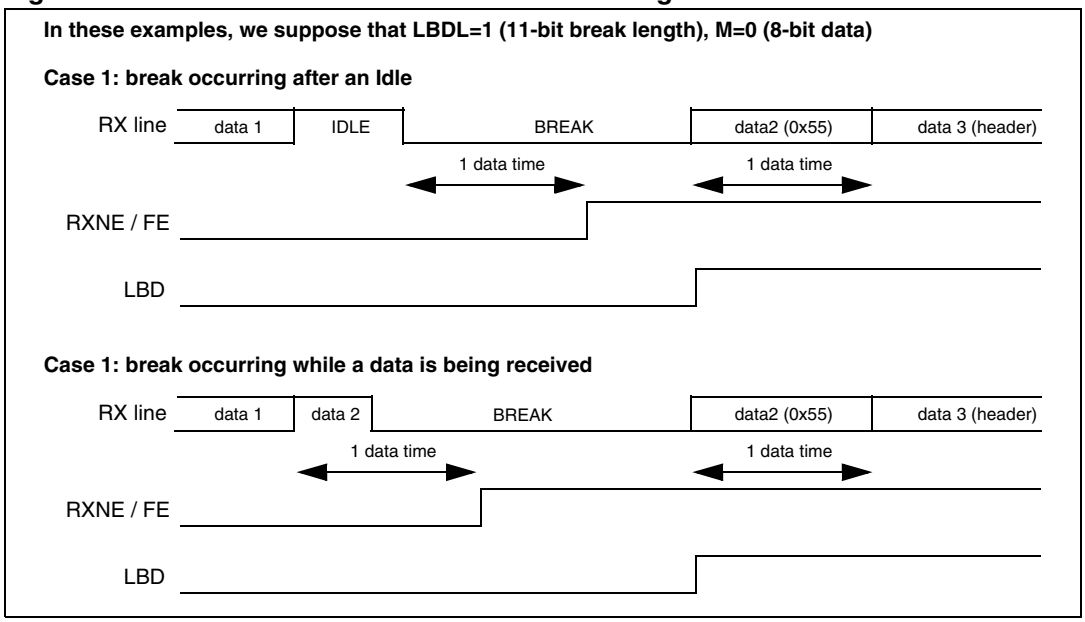


Figure 245. Break detection in LIN mode vs. Framing error detection



24.3.8 USART synchronous mode

The synchronous mode is selected by writing the CLKEN bit in the USART_CR2 register to 1. In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register,
- SCEN, HDSEL and IREN bits in the USART_CR3 register.

The USART allows the user to control a bidirectional synchronous serial communications in master mode. The SCLK pin is the output of the USART transmitter clock. No clock pulses are sent to the SCLK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART_CR2 register clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the USART_CR2 register allows the user to select the clock polarity, and the CPHA bit in the USART_CR2 register allows the user to select the phase of the external clock (see *Figure 246*, *Figure 247* & *Figure 248*).

During idle, preamble and send break, the external SCLK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as SCLK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

In this mode the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on SCLK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time must be respected (which depends on the baud rate: 1/16 bit time).

Note: 1 The SCLK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled (TE=1) and a data is being transmitted (the data register USART_DR

has been written). This means that it is not possible to receive a synchronous data without transmitting data.

2 *The LBCL, CPOL and CPHA bits have to be selected when both the transmitter and the receiver are disabled (TE=RE=0) to ensure that the clock pulses function correctly. These bits should not be changed while the transmitter or the receiver is enabled.*

3 *It is advised that TE and RE are set in the same instruction in order to minimize the setup and the hold time of the receiver.*

4 *The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).*

Figure 246. USART example of synchronous transmission

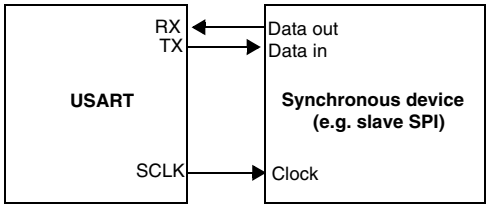


Figure 247. USART data clock timing diagram (M=0)

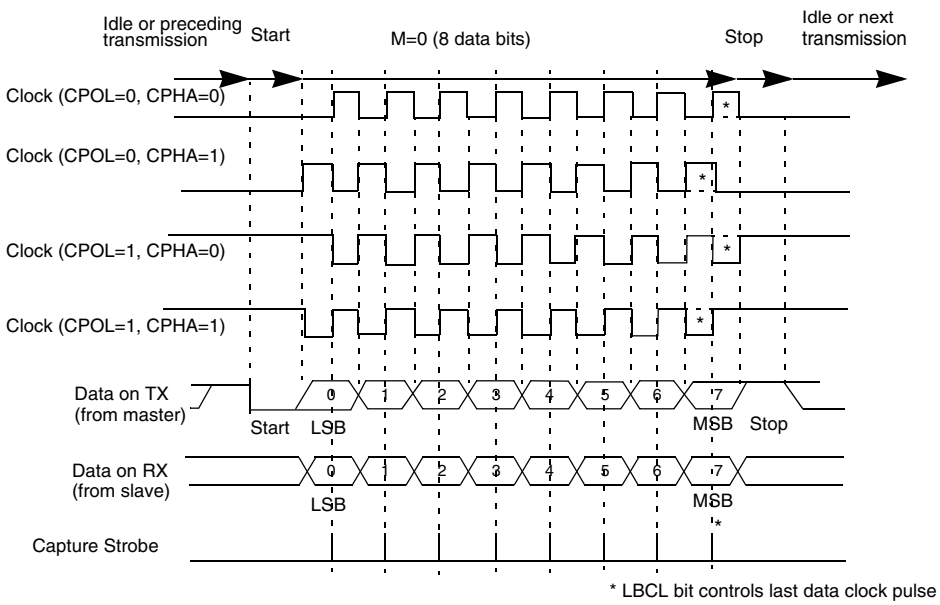


Figure 248. USART data clock timing diagram (M=1)

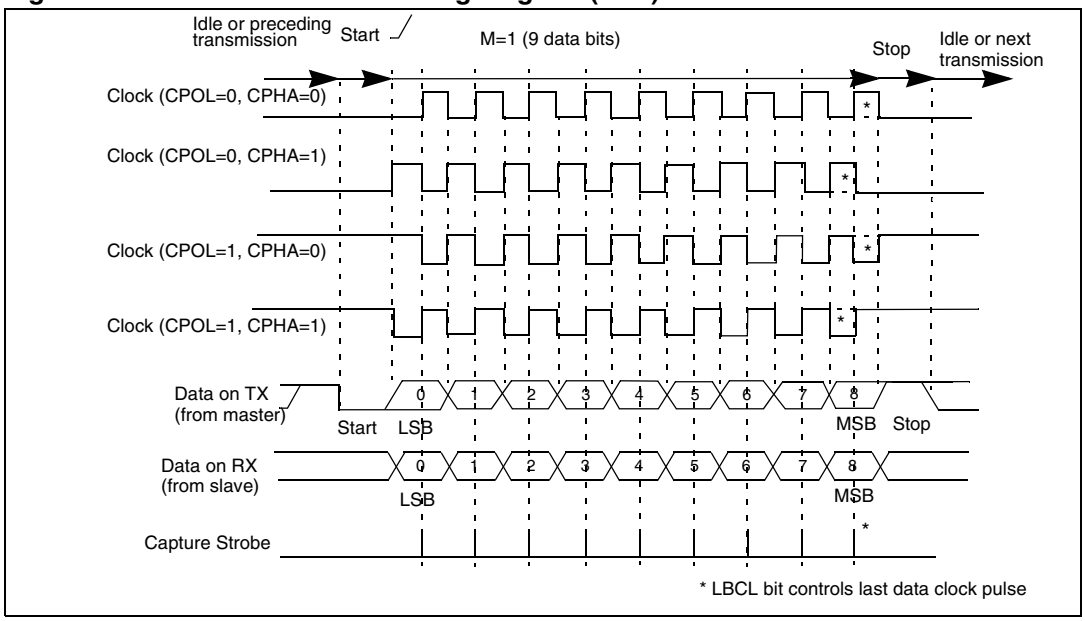
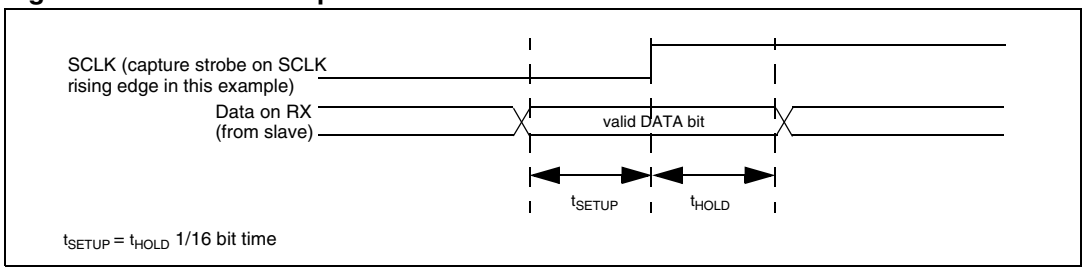


Figure 249. RX data setup/hold time



Note: The function of SCLK is different in Smartcard mode. Refer to the Smartcard mode chapter for more details.

24.3.9 Single wire half duplex communication

The single-wire half-duplex mode is selected by setting the HDSEL bit in the USART_CR3 register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART_CR2 register,
- SCEN and IREN bits in the USART_CR3 register.

The USART can be configured to follow a single wire half duplex protocol. The selection between half and full duplex communication is done with a control bit 'HALF DUPLEX SEL' (HDSEL in USART_CR3).

As soon as HDSEL is written to 1:

- RX is no longer used,
- TX is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception. It means that the I/O must be configured so that TX is configured as floating input (or output high open-drain) when not driven by the USART.

Apart from this, the communications are similar to what is done in normal USART mode. The conflicts on the line must be managed by the software (by the use of a centralized

arbiter, for instance). In particular, the transmission is never blocked by hardware and continue to occur as soon as a data is written in the data register while the TE bit is set.

24.3.10 Smartcard

The Smartcard mode is selected by setting the SCEN bit in the USART_CR3 register. In smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register,
- HDSEL and IREN bits in the USART_CR3 register.

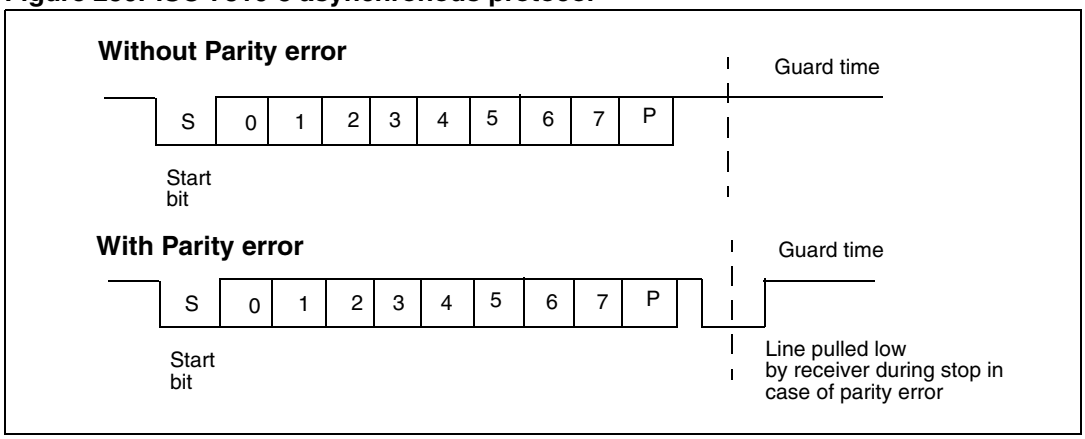
Moreover, the CLKEN bit may be set in order to provide a clock to the smartcard.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO7816-3 standard. USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register and either:
  - 0.5 stop bits when receiving: where STOP='01' in the USART_CR2 register
  - 1.5 stop bits when transmitting: where STOP='11' in the USART_CR2 register.

Figure 250 shows examples of what can be seen on the data line with and without parity error.

Figure 250. ISO 7816-3 asynchronous protocol



When connected to a smartcard, the TX output of the USART drives a bidirectional line that the smartcard also drives into. To do so, SW_RX must be connected on the same I/O than TX at product level. The Transmitter output enable TX_EN is asserted during the transmission of the start bit and the data byte, and is deasserted during the stop bit (weak pull up), so that the receive can drive the line in case of a parity error. If TX_EN is not used, TX is driven at high level during the stop bit: Thus the receiver can drive the line as long as TX is configured in open-drain.

Smartcard is a single wire half duplex communication protocol.

- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register will start shifting on the next baud clock edge. In Smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- If a parity error is detected during reception of a frame programmed with a 1/2 stop bit period, the transmit line is pulled low for a baud clock period after the completion of the receive frame, i.e. at the end of the 1/2 stop bit period. This is to indicate to the Smartcard that the data transmitted to USART has not been correctly received. This

NACK signal (pulling transmit line low for 1 baud clock) will cause a framing error on the transmitter side (configured with 1.5 stop bits). The application can handle re-sending of data according to the protocol. A parity error is 'NACK'ed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted.

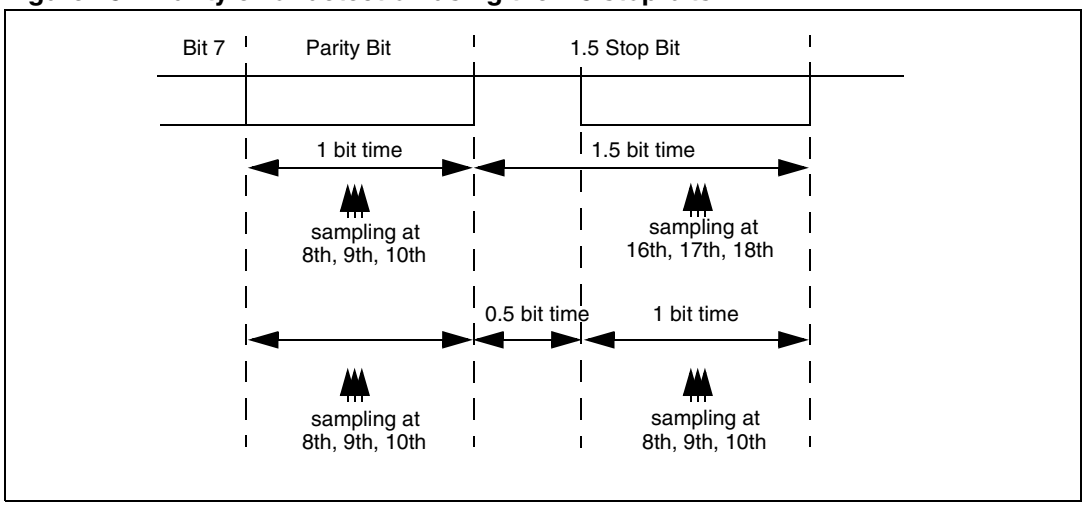
- The assertion of the TC flag can be delayed by programming the Guard Time register. In normal operation, TC is asserted when the transmit shift register is empty and no further transmit requests are outstanding. In Smartcard mode an empty transmit shift register triggers the guard time counter to count up to the programmed value in the Guard Time register. TC is forced low during this time. When the guard time counter reaches the programmed value TC is asserted high.
- The de-assertion of TC flag is unaffected by Smartcard mode.
- If a framing error is detected on the transmitter end (due to a NACK from the receiver), the NACK will not be detected as a start bit by the receive block of the transmitter. According to the ISO protocol, the duration of the received NACK can be 1 or 2 baud clock periods.
- On the receiver side, if a parity error is detected and a NACK is transmitted the receiver will not detect the NACK as a start bit.

Note: 1 A break character is not significant in Smartcard mode. A 0x00 data with a framing error will be treated as data and not as a break.

2 No IDLE frame is transmitted when toggling the TE bit. The IDLE frame (as defined for the other configurations) is not defined by the ISO protocol.

Figure 251 details how the NACK signal is sampled by the USART. In this example the USART is transmitting a data and is configured with 1.5 stop bits. The receiver part of the USART is enabled in order to check the integrity of the data and the NACK signal.

Figure 251. Parity error detection using the 1.5 stop bits



The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler. The division ratio is configured in the prescaler register USART_GTPR. SCLK frequency can be programmed from $f_{CK}/2$ to $f_{CK}/62$, where f_{CK} is the peripheral input clock.

24.3.11 IrDA SIR ENDEC block

The IrDA mode is selected by setting the IREN bit in the USART_CR3 register. In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CLKEN bits in the USART_CR2 register,
- SCEN and HDSEL bits in the USART_CR3 register.

The IrDA SIR physical layer specifies use of a Return to Zero, Inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse (see *Figure 252*).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from USART. The output pulse stream is transmitted to an external output driver and infrared LED. USART supports only bit rates up to 115.2Kbps for the SIR ENDEC. In normal mode the transmitted pulse width is specified as 3/16 of a bit period.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to USART. The decoder input is normally HIGH (marking state) in the idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (i.e. the USART is sending data to the IrDA encoder), any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy (USART is receiving decoded data from the USART), data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.
- A ‘0’ is transmitted as a high pulse and a ‘1’ is transmitted as a ‘0’. The width of the pulse is specified as 3/16th of the selected bit period in normal mode (see *Figure 253*).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for USART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when idle.
- The IrDA specification requires the acceptance of pulses greater than 1.41 us. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in the IrDA low-power Baud Register, USART_GTPR). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods will be accepted as a pulse. The IrDA encoder/decoder doesn’t work when PSC=0.
- The receiver can communicate with a low-power transmitter.
- In IrDA mode, the STOP bits in the USART_CR2 register must be configured to “1 stop bit”.

IrDA low-power mode

Transmitter:

In low-power mode the pulse width is not maintained at 3/16 of the bit period. Instead, the width of the pulse is 3 times the low-power baud rate which can be a minimum of 1.42 MHz. Generally this value is 1.8432 MHz (1.42 MHz < PSC< 2.12 MHz). A low-power mode programmable divisor divides the system clock to achieve this value.

Receiver:

Receiving in low-power mode is similar to receiving in normal mode. For glitch detection the USART should discard pulses of duration shorter than 1/PSC. A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in USART_GTPR).

Note: 1 A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.

2 The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).

Figure 252. IrDA SIR ENDEC- block diagram

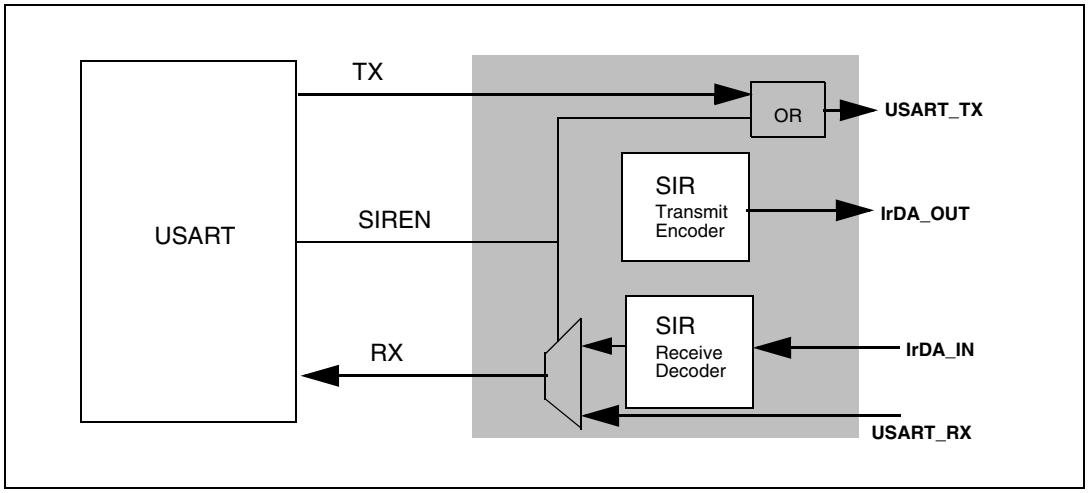
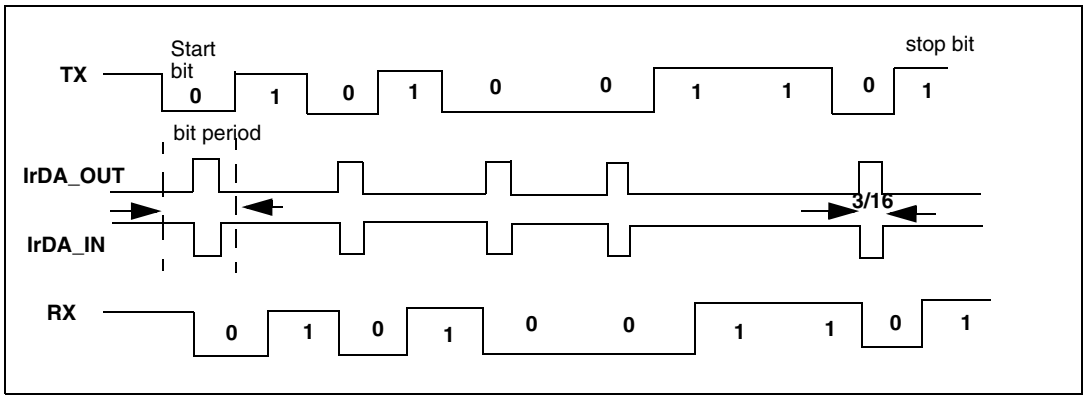


Figure 253. IrDA data modulation (3/16) -Normal Mode



24.3.12 Continuous communication using DMA

The USART is capable to continue communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

Note: You should refer to product specs for availability of the DMA controller. If DMA is not available in the product, you should use the USART as explained in Section 24.3.2 or 24.3.3. In the USART_SR register, you can clear the TXE/ RXNE flags to achieve continuous communication.

Transmission using DMA

DMA mode can be enabled for transmission by setting DMAT bit in the USART_CR3 register. Data is loaded from a SRAM area configured using the DMA peripheral (refer to the DMA specification) to the USART_DR register whenever the TXE bit is set. To map a DMA channel for USART transmission, use the following procedure (x denotes the channel number):

1. Write the USART_DR register address in the DMA control register to configure it as the destination of the transfer. The data will be moved to this address from memory after each TXE event.
2. Write the memory address in the DMA control register to configure it as the source of the transfer. The data will be loaded into the USART_DR register from this memory area after each TXE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA register
5. Configure DMA interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector. The DMAT bit should be cleared by software in the USART_CR3 register during the interrupt subroutine.

Note: If DMA is used for transmission, do not enable the TXEIE bit.

Reception using DMA

DMA mode can be enabled for reception by setting the DMAR bit in USART_CR3 register. Data is loaded from the USART_DR register to a SRAM area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure:

1. Write the USART_DR register address in the DMA control register to configure it as the source of the transfer. The data will be moved from this address to the memory after each RXNE event.
2. Write the memory address in the DMA control register to configure it as the destination of the transfer. The data will be loaded from USART_DR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred in the DMA control register.
4. Configure the channel priority in the DMA control register
5. Configure interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA control register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector. The DMAR bit should be cleared by software in the USART_CR3 register during the interrupt subroutine.

Note: If DMA is used for reception, do not enable the RXNEIE bit.

Error flagging and interrupt generation in multibuffer communication

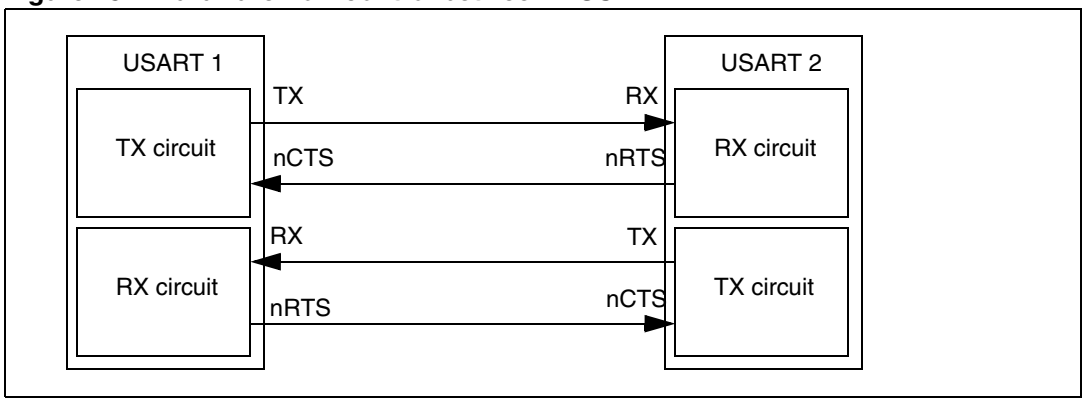
In case of multibuffer communication if any error occurs during the transaction the error flag will be asserted after the current byte. An interrupt will be generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in

case of single byte reception, there will be separate error flag interrupt enable bit (EIE bit in the USART_CR3 register), which if set will issue an interrupt after the current byte with either of these errors.

24.3.13 Hardware flow control

It is possible to control the serial data flow between 2 devices by using the nCTS input and the nRTS output. The *Figure 254* shows how to connect 2 devices in this mode:

Figure 254. Hardware flow control between 2 USART

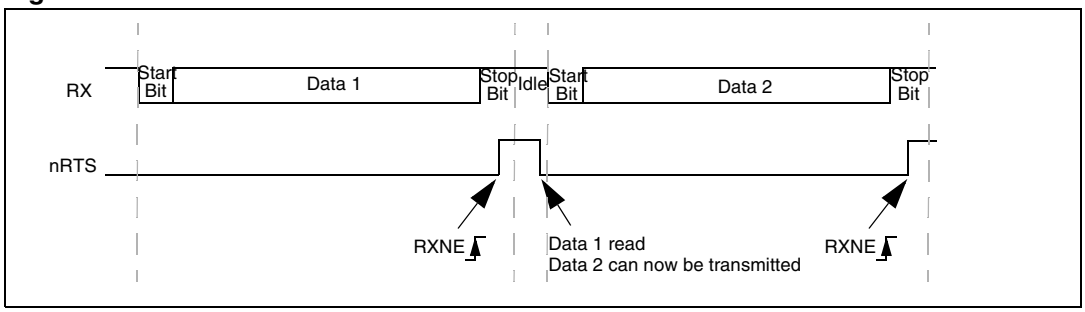


RTS and CTS flow control can be enabled independently by writing respectively RTSE and CTSE bits to 1 (in the USART_CR3 register).

RTS flow control

If the RTS flow control is enabled (RTSE=1), then nRTS is asserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register becomes empty, nRTS is deasserted, indicating that the transmission is expected to stop at the end of the current frame. *Figure 255* shows an example of communication with RTS flow control enabled.

Figure 255. RTS flow control

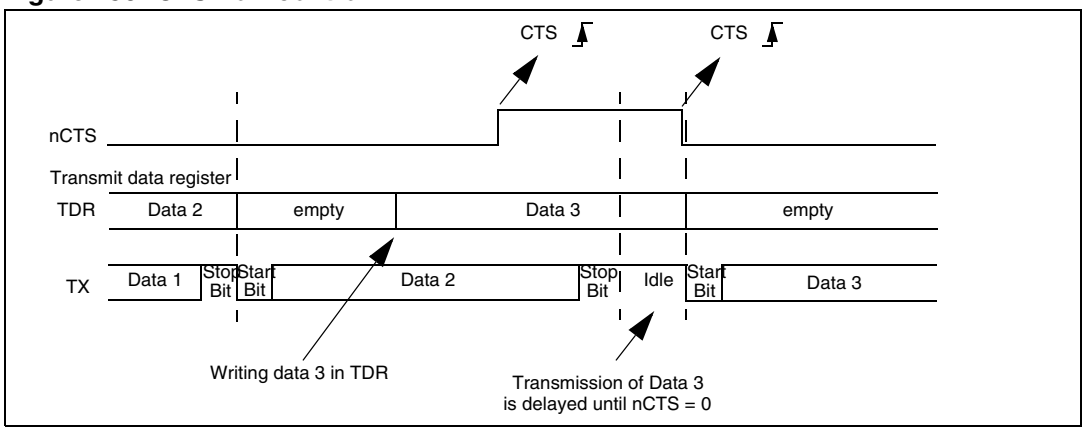


CTS flow control

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the nCTS input before transmitting the next frame. If nCTS is asserted (tied low), then the next data is transmitted (assuming that a data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When nCTS is deasserted during a transmission, the current transmission is completed before the transmitter stops.

When CTSE=1, the CTSIF status bit is automatically set by hardware as soon as the nCTS input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART_CR3 register is set. The figure below shows an example of communication with CTS flow control enabled.

Figure 256. CTS flow control



24.4 USART interrupts

Table 156. USART interrupt requests

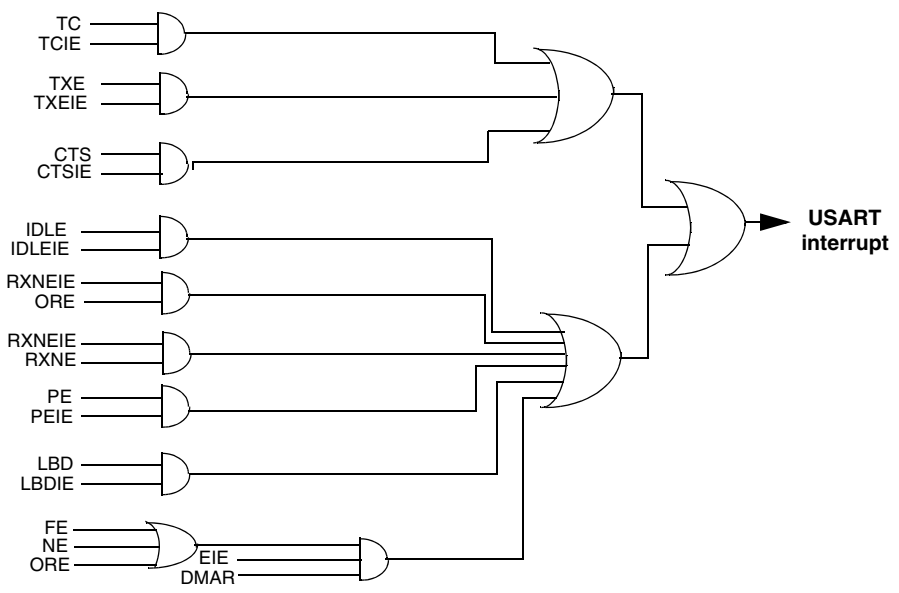
Interrupt event	Event flag	Enable Control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication	NE or ORE or FE	EIE

The USART interrupt events are connected to the same interrupt vector (see *Figure 257*).

- During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.
- While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 257. USART interrupt mapping diagram



24.5 USART mode configuration

Table 157. USART modes configuration(1)

USART modes	USART1	USART2	USART3	UART4	UART5
Asynchronous mode	X	X	X	X	X
Hardware Flow Control	X	X	X	NA	NA
Multibuffer Communication (DMA)	X	X	X	X	NA
Multiprocessor Communication	X	X	X	X	X
Synchronous	X	X	X	NA	NA
Smartcard	X	X	X	NA	NA
Half-Duplex (Single-Wire mode)	X	X	X	X	X
IrDA	X	X	X	X	X
LIN	X	X	X	X	X

1. X = supported; NA = not applicable.

24.6 USART registers

Refer to *Section 1.1 on page 32* for a list of abbreviations used in register descriptions.

24.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
Res.						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Bits 31:10 Reserved, forced by hardware to 0.

Bit 9 **CTS**: *CTS Flag*

This bit is set by hardware when the nCTS input toggles, if the CTSE bit is set. It is cleared by software (by writing it to 0). An interrupt is generated if CTSIE=1 in the USART_CR3 register.

0: No change occurred on the nCTS status line

1: A change occurred on the nCTS status line

Note: This bit is not available for UART4 & UART5.

Bit 8 **LBD**: *LIN Break Detection Flag*

This bit is set by hardware when the LIN break is detected. It is cleared by software (by writing it to 0). An interrupt is generated if LBDIE = 1 in the USART_CR2 register.

0: LIN Break not detected

1: LIN break detected

Note: An interrupt is generated when LBD=1 if LBDIE=1

Bit 7 **TXE**: *Transmit Data Register Empty*

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register

1: Data is transferred to the shift register)

Note: This bit is used during single buffer transmission.

Bit 6 **TC**: *Transmission Complete.*

This bit is set by hardware when transmission of a frame containing Data is complete. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a write to the USART_DR register). The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for multibuffer communication.

0: Transmission is not complete

1: Transmission is complete

Bit 5 **RXNE**: *Read Data Register Not Empty.*

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_DR register. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. This clearing sequence is recommended only for multibuffer communication.

0: Data is not received

1: Received data is ready to be read.

Bit 4 **IDLE**: *IDLE line detected.*

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the IDLEIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Idle Line is detected

1: Idle Line is detected

Note: The IDLE bit will not be set again until the RXNE bit has been set itself (i.e. a new idle line occurs).

Bit 3 **ORE**: *OverRun Error.*

This bit is set by hardware when the word currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Overrun error

1: Overrun error is detected

Note: When this bit is set, the RDR register content will not be lost but the shift register will be overwritten. An interrupt is generated on ORE flag in case of Multi Buffer communication if the EIE bit is set.

Bit 2 **NE**: *Noise Error Flag.*

This bit is set by hardware when noise is detected on a received frame. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No noise is detected

1: Noise is detected

Note: This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupting interrupt is generated on NE flag in case of Multi Buffer communication if the EIE bit is set.

Bit 1 **FE**: *Framing Error.*

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Framing error is detected

1: Framing error or break character is detected

Note: This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. If the word currently being transferred causes both frame error and overrun error, it will be transferred and only the ORE bit will be set.

An interrupt is generated on FE flag in case of Multi Buffer communication if the EIE bit is set.

Bit 0 **PE**: *Parity Error.*

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by a software sequence (a read to the status register followed by a read to the USART_DR data register). The software must wait for the RXNE flag to be set before clearing the PE bit.

An interrupt is generated if PEIE = 1 in the USART_CR1 register.

0: No parity error

1: Parity error

24.6.2 Data register (USART_DR)

Address offset: 0x04

Reset value: Undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DR[8:0]								
Res.							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:9 Reserved, forced by hardware to 0.

Bits 8:0 **DR[8:0]**: *Data value.*

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting with the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

24.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: *mantissa of USARTDIV.*

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: *fraction of USARTDIV.*

These 4 bits define the fraction of the USART Divider (USARTDIV)

24.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
Res.		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:14 Reserved, forced by hardware to 0.

Bit 13 **UE**: *USART Enable.*

When this bit is cleared the USART prescalers and outputs are stopped and the end of the current

byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: USART prescaler and outputs disabled

1: USART enabled

Bit 12 **M**: *word length.*

This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, 1 Stop bit

Note:

The M bit must not be modified during a data transfer (both transmission and reception)

Bit 11 **WAKE**: *Wakeup method.*

This bit determines the USART wakeup method, it is set or cleared by software.

0: Idle Line

1: Address Mark

Bit 10 **PCE**: *Parity Control Enable.*

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

Bit 9 **PS**: *Parity Selection.*

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity will be selected after the current byte.

0: Even parity

1: Odd parity

Bit 8 **PEIE**: *PE Interrupt Enable.*

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever PE=1 in the USART_SR register

Bit 7 **TXEIE**: *TXE Interrupt Enable.*

This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever TXE=1 in the USART_SR register

Bit 6 **TCIE**: *Transmission Complete Interrupt Enable.*

This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever TC=1 in the USART_SR register

Bit 5 **RXNEIE**: *RXNE Interrupt Enable.*

This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_SR register

Bit 4 **IDLEIE**: *IDLE Interrupt Enable.*

This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever IDLE=1 in the USART_SR register

Bit 3 **TE**: *Transmitter Enable.*

This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled

Notes:

1: During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble (idle line) after the current word, except in smartcard mode.

2: When TE is set there is a 1 bit-time delay before the transmission starts.

Bit 2 **RE**: *Receiver Enable.*

This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit

Bit 1 **RWU**: *Receiver wakeup.*

This bit determines if the USART is in mute mode or not. It is set and cleared by software and can be cleared by hardware when a wakeup sequence is recognized.
0: Receiver in active mode
1: Receiver in mute mode

Notes:

1: Before selecting Mute mode (by setting the RWU bit) the USART must first receive a data byte, otherwise it cannot function in Mute mode with wakeup by Idle line detection.

2: In Address Mark Detection wakeup configuration (WAKE bit=1) the RWU bit cannot be modified by software while the RXNE bit is set.

Bit 0 **SBK**: *Send Break.*

This bit set is used to send break characters. It can be set and cleared by software. It should be set by software, and will be reset by hardware during the stop bit of break.
0: No break character is transmitted
1: Break character will be transmitted

24.6.5 Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, forced by hardware to 0.

Bit 14 **LINEN**: *LIN mode enable*

This bit is set and cleared by software.

0: LIN mode disabled

1: LIN mode enabled

The LIN mode enables the capability to send LIN Synch Breaks (13 low bits) using the SBK bit in the USART_CR1 register, and to detect LIN Sync breaks.

Bits 13:12 **STOP**: *STOP bits.*

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

Note: The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.

Bit 11 **CLKEN**: *Clock Enable.*

This bit allows the user to enable the SCLK pin.

0: SCLK pin disabled

1: SCLK pin enabled

Note: This bit is not available for UART4 & UART5.

Bit 10 **CPOL**: *Clock Polarity.*

This bit allows the user to select the polarity of the clock output on the SCLK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: Steady low value on SCLK pin outside transmission window.

1: Steady high value on SCLK pin outside transmission window.

Note: This bit is not available for UART4 & UART5.

Bit 9 **CPHA**: *Clock Phase*

This bit allows the user to select the phase of the clock output on the SCLK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see figures *247* to *248*)

0: The first clock transition is the first data capture edge.

1: The second clock transition is the first data capture edge.

Note: This bit is not available for UART4 & UART5.

Bit 8 **LBCL**: *Last Bit Clock pulse.*

This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the SCLK pin in synchronous mode.

0: The clock pulse of the last data bit is not output to the SCLK pin.

1: The clock pulse of the last data bit is output to the SCLK pin.

Note: 1: The last bit is the 8th or 9th data bit transmitted depending on the 8 or 9 bit format selected by the M bit in the USART_CR1 register.

2: This bit is not available for UART4 & UART5.

Bit 7 Reserved, forced by hardware to 0.

Bit 6 **LBDIE**: *LIN Break Detection Interrupt Enable.*

Break interrupt mask (break detection using break delimiter).

0: Interrupt is inhibited

1: An interrupt is generated whenever LBD=1 in the USART_SR register

Bit 5 **LBDL**: *LIN Break Detection Length.*

This bit is for selection between 11 bit or 10 bit break detection.

0: 10 bit break detection

1: 11 bit break detection

Bit 4 Reserved, forced by hardware to 0.

Bits 3:0 **ADD[3:0]**: *Address of the USART node.*

This bit-field gives the address of the USART node.

This is used in multiprocessor communication during mute mode, for wake up with address mark detection.

Note: These 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.

24.6.6 Control register 3 (USART_CR3)

Address offset: 0x14

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HD SEL	IRLP	IREN	EIE
Res.					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:11 Reserved, forced by hardware to 0.

Bit 10 **CTSIE**: *CTS Interrupt Enable.*

0: Interrupt is inhibited

1: An interrupt is generated whenever CTS=1 in the USART_SR register

Note: This bit is not available for UART4 & UART5.

Bit 9 **CTSE**: *CTS Enable.*

0: CTS hardware flow control disabled

1: CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0). If the nCTS input is deasserted while a data is being transmitted, then the transmission is completed before stopping. If a data is written into the data register while nCTS is asserted, the transmission is postponed until nCTS is asserted.

Note: This bit is not available for UART4 & UART5.

Bit 8 **RTSE**: *RTS Enable.*

0: RTS hardware flow control disabled

1: RTS interrupt enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The nRTS output is asserted (tied to 0) when a data can be received.

Note: This bit is not available for UART4 & UART5.

Bit 7 **DMAT**: *DMA Enable Transmitter.*

This bit is set/reset by software

1: DMA mode is enabled for transmission.

0: DMA mode is disabled for transmission.

Note: This bit is not available for UART5.

Bit 6 **DMAR**: *DMA Enable Receiver.*

This bit is set/reset by software

1: DMA mode is enabled for reception.

0: DMA mode is disabled for reception.

Note: This bit is not available for UART5.

Bit 5 **SCEN**: *Smartcard mode enable.*

This bit is used for enabling Smartcard mode.

0: Smartcard Mode disabled

1: Smartcard Mode enabled

Note: This bit is not available for UART4 & UART5.

Bit 4 **NACK**: *Smartcard NACK enable.*

0: NACK transmission in case of parity error is disabled

1: NACK transmission during parity error is enabled.

Note: This bit is not available for UART4 & UART5.

Bit 3 **HDSEL**: *Half-Duplex Selection.*

Selection of Single-wire Half-duplex mode

0: Half duplex mode is not selected

1: Half duplex mode is selected

Bit 2 **IRLP**: *IrDA Low-Power.*

This bit is used for selecting between normal and low-power IrDA modes

0: Normal mode

1: Low-power mode

Bit 1 **IREN**: *IrDA mode Enable.*

This bit is set and cleared by software.

0: IrDA disabled

1: IrDA enabled

Bit 0 **EIE**: *Error Interrupt Enable.*

Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing error, overrun error or noise error (FE=1 or ORE=1 or NE=1 in the USART_SR register) in case of Multi Buffer Communication (DMAR=1 in the USART_CR3 register).

0: Interrupt is inhibited

1: An interrupt is generated whenever DMAR=1 in the USART_CR3 register and FE=1 or ORE=1 or NE=1 in the USART_SR register.

24.6.7 Guard time and prescaler register (USART_GTPR)

Address offset: 0x18

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GT[7:0]								PSC[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:8 **GT[7:0]**: *Guard time value.*

This bit-field gives the Guard time value in terms of number of baud clocks.

This is used in Smartcard mode. The Transmission Complete flag is set after this guard time value.

Note: This bit is not available for UART4 & UART5.

Bits 7:0 **PSC[7:0]**: *Prescaler value.*

– **In IrDA Low-power mode:**

PSC[7:0] = IrDA Low-Power Baud Rate

Used for programming the prescaler for dividing the system clock to achieve the low-power frequency:

The source clock is divided by the value given in the register (8 significant bits):

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

– **In normal IrDA mode:** PSC must be set to 00000001.

– **In smartcard mode:**

PSC[4:0]: *Prescaler value.*

Used for programming the prescaler for dividing the system clock to provide the smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

00000: Reserved - do not program this value

00001: divides the source clock by 2

00010: divides the source clock by 4

00011: divides the source clock by 6

...

Note: 1: Bits [7:5] have no effect if Smartcard mode is used.

2: This bit is not available for UART4 & UART5.

24.6.8 USART register map

The table below gives the USART register map and reset values.

Table 158. USART register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																							DR[8:0]								
	Reset value																								0	0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to *Table 1 on page 35* for the register boundary addresses.

25 Device electronic signature

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

The electronic signature is stored in the System memory area in the Flash memory module, and can be read using the JTAG/SWD or the CPU. It contains factory-programmed identification data that allow the user firmware or other external devices to automatically match its interface to the characteristics of the STM32F10xxx microcontroller.

25.1 Memory size registers

25.1.1 Flash size register

Base address: 0x1FFF F7E0

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F_SIZE															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **F_SIZE:** *Flash memory size*

This field value indicates the Flash memory size of the device in Kbytes.
Example: 0x0080 = 128 Kbytes.

25.2 Unique device ID register (96 bits)

The unique device identifier is ideally suited:

- for use as serial numbers (for example USB string serial numbers or other end applications)
- for use as security keys in order to increase the security of code in Flash memory while using and combining this unique ID with software cryptographic primitives and protocols before programming the internal Flash memory
- to activate secure boot processes, etc.

The 96-bit unique device identifier provides a reference number which is unique for any device and in any context. These bits can never be altered by the user.

The 96-bit unique device identifier can also be read in single bytes/half-words/words in different ways and then be concatenated using a custom algorithm.

Base address: 0x1FFF F7E8

Address offset: 0x00

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U_ID(15:0)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **U_ID(15:0):** *15:0 unique ID bits.*

Address offset: 0x02

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U_ID(31:16)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **U_ID(31:16):** *31:16 unique ID bits.*
This field value is also reserved for a future feature.

Address offset: 0x04

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
U_ID(63:48)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U_ID(47:32)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **U_ID(63:32):** *63:32 unique ID bits.*

Address offset: 0x08

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
U_ID(95:80)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U_ID(79:64)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **U_ID(95:64):** *95:64 Unique ID bits.*

26 Debug support (DBG)

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 64 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

This Section applies to the whole STM32F10xxx family, unless otherwise specified.

26.1 Overview

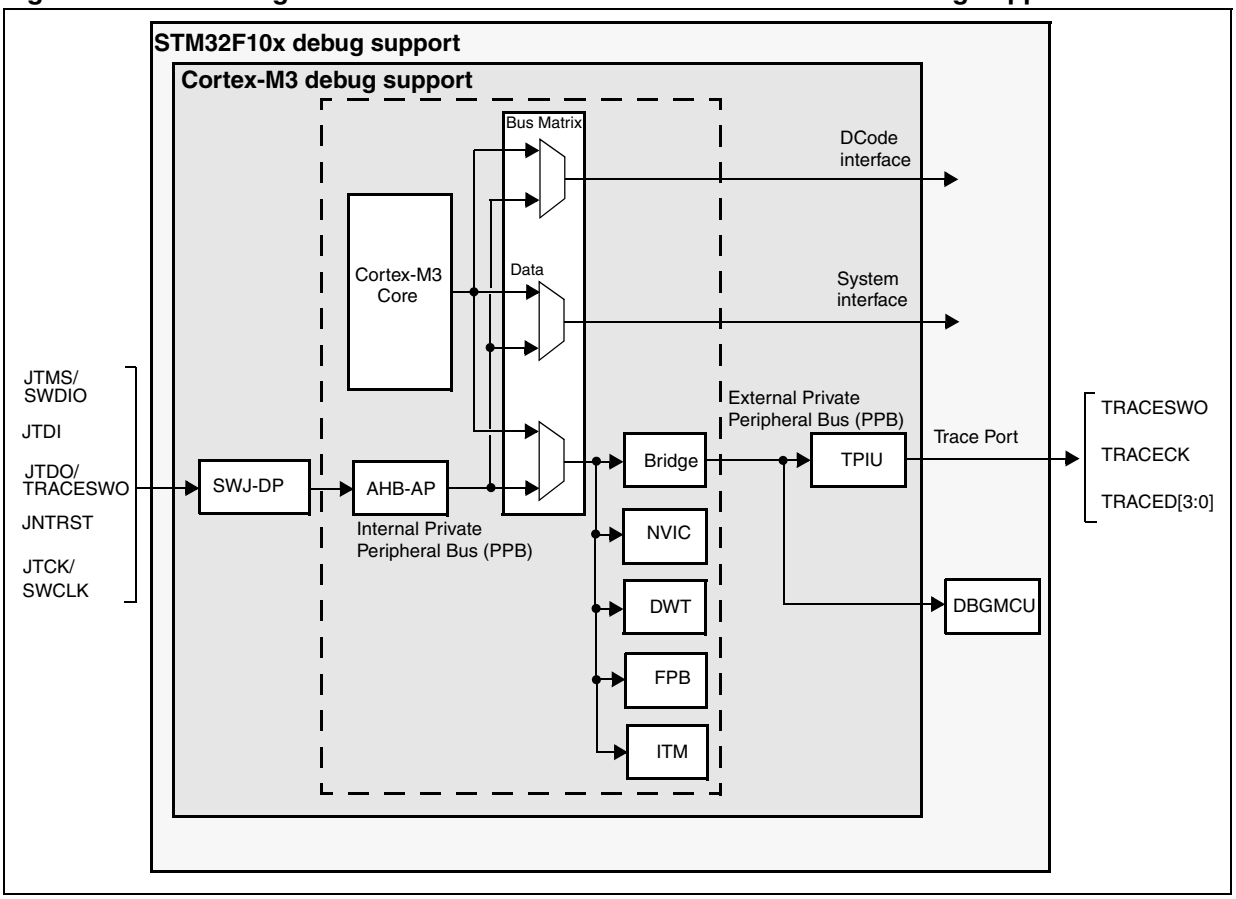
The STM32F10xxx is built around a Cortex-M3 core which contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When stopped, the core's internal state and the system's external state may be examined. Once examination is complete, the core and the system may be restored and program execution resumed.

The debug features are used by the debugger host when connecting to and debugging the STM32F10xxx MCU.

Two interfaces for debug are available:

- Serial wire
- JTAG debug port

Figure 258. Block diagram of STM32F10xxx-level and Cortex-M3-level debug support



Note: The debug features embedded in the Cortex-M3 core are a subset of the ARM CoreSight Design Kit.

The ARM Cortex-M3 core provides integrated on-chip debug support. It is comprised of:

- SWJ-DP: Serial wire / JTAG debug port
- AHP-AP: AHB access port
- ITM: Instrumentation trace macrocell
- FPB: Flash patch breakpoint
- DWT: Data watchpoint trigger
- TPUI: Trace port unit interface (available on larger packages, where the corresponding pins are mapped)

It also includes debug features dedicated to STM32F10xxx:

- Flexible debug pinout assignment
- MCU debug box (support for low-power modes, control over peripheral clocks, etc.)

Note: For further information on debug functionality supported by the ARM Cortex-M3 core, refer to the Cortex-M3 r1p1 Technical Reference Manual (see Related documents on page 1) and to the CoreSight Design Kit r1p0 TRM.

26.2 Reference ARM documentation

- Cortex™-M3 r1p1 Technical Reference Manual (TRM) *(see Related documents on page 1)*
- ARM Debug Interface V5
- ARM CoreSight Design Kit revision r1p0 Technical Reference Manual

26.3 SWJ debug port (serial wire and JTAG)

The STM32F10xxx core integrates the Serial Wire / JTAG Debug Port (SWJ-DP). It is an ARM standard CoreSight debug port that combines a JTAG-DP (5-pin) interface and a SW-DP (2-pin) interface.

- The JTAG Debug Port (JTAG-DP) provides a 5-pin standard JTAG interface to the AHP-AP port.
- The Serial Wire Debug Port (SW-DP) provides a 2-pin (clock + data) interface to the AHP-AP port.

In the SWJ-DP, the two JTAG pins of the SW-DP are multiplexed with some of the five JTAG pins of the JTAG-DP.

Figure 259. SWJ debug port

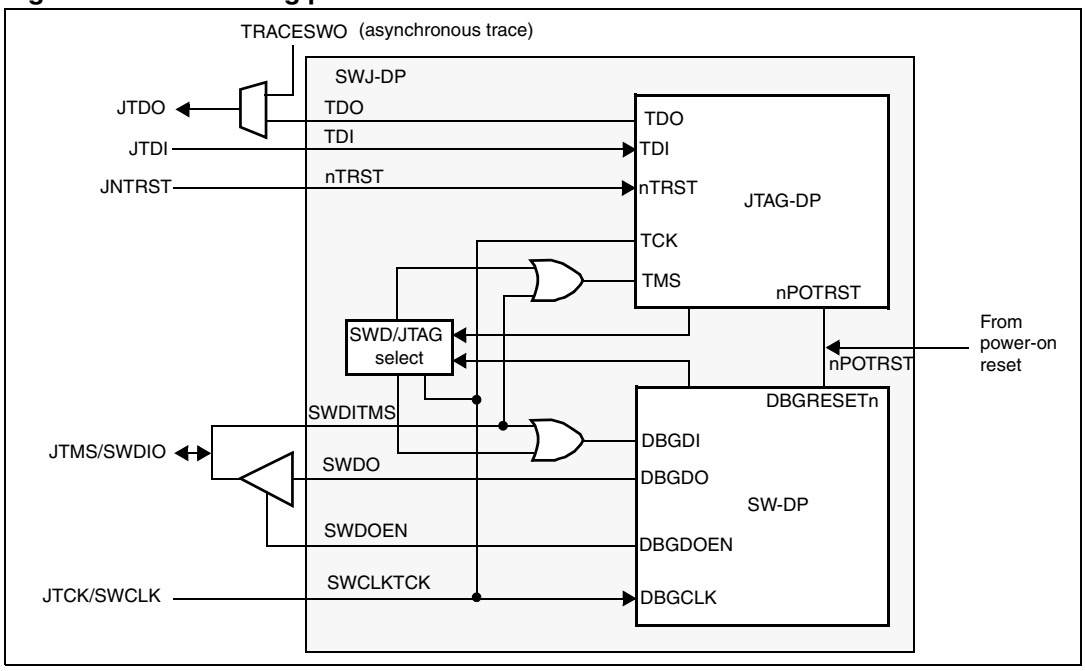


Figure 259 shows that the asynchronous TRACE output (TRACESWO) is multiplexed with TDO. This means that the asynchronous trace can only be used with SW-DP, not JTAG-DP.

26.3.1 Mechanism to select the JTAG-DP or the SW-DP

By default, the JTAG-Debug Port is active.

If the debugger host wants to switch to the SW-DP, it must provide a dedicated JTAG sequence on TMS/TCK (respectively mapped to SWDIO and SWCLK) which disables the

JTAG-DP and enables the SW-DP. This way it is possible to activate the SWDP using only the SWCLK and SWDIO pins.

This sequence is:

1. Send more than 50 TCK cycles with TMS (SWDIO) =1
2. Send the 16-bit sequence on TMS (SWDIO) = 0111100111100111 (MSB transmitted first)
3. Send more than 50 TCK cycles with TMS (SWDIO) =1

26.4 Pinout and debug port pins

The STM32F10xxx MCU is available in various packages with different numbers of available pins. As a result, some functionality related to pin availability may differ between packages.

26.4.1 SWJ debug port pins

Five pins are used as outputs from the STM32F10xxx for the SWJ-DP as *alternate functions* of General Purpose I/Os. These pins are available on all packages.

Table 159. SWJ debug port pins

SWJ-DP pin name	JTAG debug port		SW debug port		Pin assign ment
	Type	Description	Type	Debug assignment	
JTMS/SWDIO	I	JTAG Test Mode Selection	I/O	Serial Wire Data Input/Output	PA13
JTCK/SWCLK	I	JTAG Test Clock	I	Serial Wire Clock	PA14
JTDI	I	JTAG Test Data Input	-	-	PA15
JTDO/TRACESWO	O	JTAG Test Data Output	-	TRACESWO if async trace is enabled	PB3
JNTRST	I	JTAG Test nReset	-	-	PB4

26.4.2 Flexible SWJ-DP pin assignment

After RESET (SYSRESETn or PORESETn), all five pins used for the SWJ-DP are assigned as dedicated pins immediately usable by the debugger host (note that the trace outputs are not assigned except if explicitly programmed by the debugger host).

However, the STM32F10xxx MCU implements the REMAP_DBGAFR register to disable some part or all of the SWJ-DP port and so releases the associated pins for General Purpose I/Os usage. This register is mapped on an APB bridge connected to the Cortex-M3 System Bus. Programming of this register is done by the user software program and not the debugger host.

Three control bits allow the configuration of the SWJ-DP pin assignments. These bits are reset by the System Reset.

- REMAP_AF_REG (@ 0x4001 0004 in STM32F10xxx MCU)
  - READ: APB - No Wait State
  - WRITE: APB - 1 Wait State if the write buffer of the AHB-APB bridge is full.

  Bit 26:24= **SWJ_CFG[2:0]**

  Set and cleared by software.

  These bits are used to configure the number of pins assigned to the SWJ debug port. The goal is to release as much as possible the number of pins to be used as General Purpose I/Os if using a small size for the debug port.

  The default state after reset is “000” (whole pins assigned for a full JTAG-DP connection). Only one of the 3 bits can be set (it is forbidden to set more than one bit).

Table 160. Flexible SWJ-DP pin assignment

SWJ_ CFG [2:0]	Available debug ports	SWJ I/O pin assigned				
		PA13 / JTMS/ SWDIO	PA14 / JTCK/ SWCLK	PA15 / JTDI	PB3 / JTDO	PB4/ JNTRST
000	Full SWJ (JTAG-DP + SW-DP) - Reset State	X	X	X	X	X
001	Full SWJ (JTAG-DP + SW-DP) but without JNTRST	X	X	X	X	
010	JTAG-DP Disabled and SW-DP Enabled	X	X			
100	JTAG-DP Disabled and SW-DP Disabled	Released				
other	Forbidden					

Note: When the APB bridge write buffer is full, it takes one extra APB cycle when writing the REMAP_AF register. This is because the deactivation of the JTAGSW pins is done in two cycles to guarantee a clean level on the nTRST and TCK input signals of the core.

- Cycle 1: the JTAGSW input signals to the core are tied to 1 or 0 (to 1 for nTRST, TDI and TMS, to 0 for TCK)
- Cycle 2: the GPI/O controller takes the control signals of the SWJTAG I/O pins (like controls of direction, pull-up/down, Schmitt trigger activation, etc.).

26.4.3 Internal pull-up and pull-down on JTAG pins

It is necessary to ensure that the JTAG input pins are not floating since they are directly connected to flip-flops to control the debug mode features. Special care must be taken with the SWCLK/TCK pin which is directly connected to the clock of some of these flip-flops.

To avoid any uncontrolled I/O levels, the STM32F10xxx embeds internal pull-ups and pull-downs on JTAG input pins:

- JNTRST: Internal pull-up
- JTDI: Internal pull-up
- JTMS/SWDIO: Internal pull-up
- TCK/SWCLK: Internal pull-down

Once a JTAG I/O is released by the user software, the GPIO controller takes control again. The reset states of the GPIO control registers put the I/Os in the equivalent state:

- JNTRST: Input pull-up
- JTDI: Input pull-up
- JTMS/SWDIO: Input pull-up
- JTCK/SWCLK: Input pull-down
- JTDO: Input floating

The software can then use these I/Os as standard GPIOs.

Note: The JTAG IEEE standard recommends to add pull-ups on TDI, TMS and nTRST but there is no special recommendation for TCK. However, for STM32F10xxx, an integrated pull-down is used for JTCK.

Having embedded pull-ups and pull-downs removes the need to add external resistors.

26.4.4 Using serial wire and releasing the unused debug pins as GPIOs

To use the serial wire DP to release some GPIOs, the user software must set SWJ_CFG=010 just after reset. This release PA15, PB3 and PB4 which now become available as GPIOs.

When debugging, the host performs the following actions:

- Under system RESET, all SWJ pins are assigned (JTAG-DP + SW-DP)
- Under system RESET, the debugger host sends the JTAG sequence to switch from the JTAG-DP to the SW-DP.
- Still under system RESET, the debugger sets a breakpoint on vector reset
- The System Reset is released and the Core halts.
- All the debug communications from this point are done using the SW-DP. The other JTAG pins can then be reassigned as GPIOs by the user software.

Note: For user software designs, note that:

To release the debug pins, remember that they will be first configured either in input-pull-up (nTRST, TMS, TDI) or pull-down (TCK) or output tristate (TDO) for a certain duration after reset until the instant when the user software releases the pins.

When debug pins (JTAG or SW or TRACE) are mapped, changing the corresponding I/O pin configuration in the IOPORT controller has no effect.

26.5 STM32F10xxx JTAG TAP connection

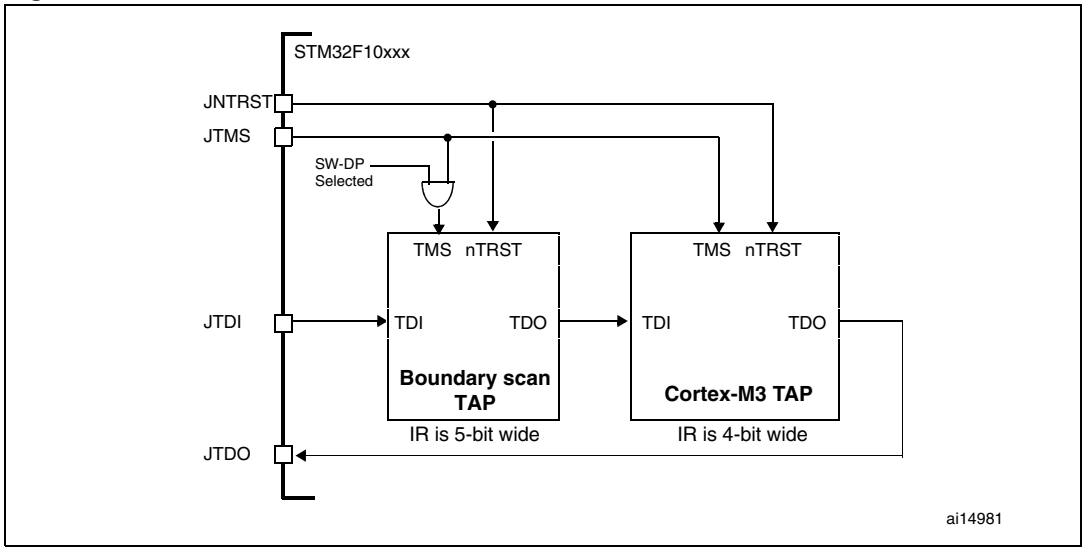
The STM32F10xxx MCU integrates two serially connected JTAG TAPs, the boundary scan TAP (IR is 5-bit wide) and the Cortex-M3 TAP (IR is 4-bit wide).

To access the TAP of the Cortex-M3 for debug purposes:

1. First, it is necessary to shift the BYPASS instruction of the boundary scan TAP.
2. Then, for each IR shift, the scan chain contains 9 bits (=5+4) and the unused TAP instruction must be shifted in using the BYPASS instruction.
3. For each data shift, the unused TAP, which is in BYPASS mode, adds 1 extra data bit in the data scan chain.

Note: ***Important****: Once Serial-Wire is selected using the dedicated ARM JTAG sequence, the boundary scan TAP is automatically disabled (JTMS forced high).*

Figure 260. JTAG TAP connections



26.6 ID codes and locking mechanism

There are several ID codes inside the STM32F10xxx MCU. ST strongly recommends tools designers to lock their debuggers using the MCU DEVICE ID code located in the external PPB memory map at address 0xE0042000.

26.6.1 MCU device ID code

The STM32F10xxx MCU integrates an MCU ID code. This ID identifies the ST MCU part-number and the die revision. It is part of the DBG_MCU component and is mapped on the external PPB bus (see *Section 26.15 on page 657*). This code is accessible using the JTAG debug port (4 to 5 pins) or the SW debug port (two pins) or by the user software. It is even accessible while the MCU is under system reset.

DBGMCU_IDCODE

Address: 0xE004 2000

Only 32-bits access supported. Read-only.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV_ID															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DEV_ID											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **REV_ID(15:0)** Revision identifier

This field indicates the revision of the device:

In low-density devices:

– 0x1000 = Revision A

In medium-density devices:

– 0x0000 = Revision A

– 0x2000 = Revision B

– 0x2001 = Revision Z

– 0x2003 = Revision Y

In high-density devices:

– 0x1000 = Revision A

– 0x1001 = Revision Z

In connectivity line devices:

– 0x1000 = Revision A

Bits 27:12 Reserved

Bits 11:0 **DEV_ID(11:0)**: Device identifier

This field indicates the device ID.

For low-density devices, the device ID is 0x412

For medium-density devices, the device ID is 0x410

For high-density devices, the device ID is 0x414

For connectivity devices, the device ID is 0x418

26.6.2 Boundary scan TAP

JTAG ID code

The TAP of the STM32F10xxx BSC (boundary scan) integrates a JTAG ID code equal to:

- In low-density devices:
  - 0x06412041 = Revision A
- In medium-density devices:
  - 0x06410041 = Revision A
  - 0x16410041 = Revision B and Revision Z
- In high-density devices:
  - 0x06414041 = Revision A

26.6.3 Cortex-M3 TAP

The TAP of the ARM Cortex-M3 integrates a JTAG ID code. This ID code is the ARM default one and has not been modified. This code is only accessible by the JTAG Debug Port. This code is **0x3BA00477** (corresponds to Cortex-M3 r1p1-01rel0, *see Related documents on page 1*).

Only the DEV_ID(11:0) should be used for identification by the debugger/programmer tools.

26.6.4 Cortex-M3 JEDEC-106 ID code

The ARM Cortex-M3 integrates a JEDEC-106 ID code. It is located in the 4KB ROM table mapped on the internal PPB bus at address 0xE00FF000_0xE00FFFFF.

This code is accessible by the JTAG Debug Port (4 to 5 pins) or by the SW Debug Port (two pins) or by the user software.

26.7 JTAG debug port

A standard JTAG state machine is implemented with a 4-bit instruction register (IR) and five data registers (for full details, refer to the *Cortex-M3 r1p1 Technical Reference Manual (TRM), for references, please see Related documents on page 1)*:

Table 161. JTAG debug port data registers

IR(3:0)	Data register	Details
1111	BYPASS [1 bit]	
1110	IDCODE [32 bits]	*ID CODE* 0x3BA00477 (ARM Cortex-M3 r1p1-01rel0 ID Code)
1010	DPACC [35 bits]	*Debug Port Access Register* This initiates a debug port and allows access to a debug port register. – When transferring data IN: Bits 34:3 = DATA[31:0] = 32-bit data to transfer for a write request Bits 2:1 = A[3:2] = 2-bit address of a debug port register. Bit 0 = RnW = Read request (1) or write request (0). – When transferring data OUT: Bits 34:3 = DATA[31:0] = 32-bit data which is read following a read request Bits 2:0 = ACK[2:0] = 3-bit Acknowledge: 010 = OK/FAULT 001 = WAIT OTHER = reserved Refer to *Table 162* for a description of the A(3:2) bits

Table 161. JTAG debug port data registers

IR(3:0)	Data register	Details
1011	APACC [35 bits]	*Access Port Access Register* Initiates an access port and allows access to an access port register. – When transferring data IN: Bits 34:3 = DATA[31:0] = 32-bit data to shift in for a write request Bits 2:1 = A[3:2] = 2-bit address (sub-address AP registers). Bit 0 = RnW= Read request (1) or write request (0). – When transferring data OUT: Bits 34:3 = DATA[31:0] = 32-bit data which is read following a read request Bits 2:0 = ACK[2:0] = 3-bit Acknowledge: 010 = OK/FAULT 001 = WAIT OTHER = reserved There are many AP Registers (see AHB-AP) addressed as the combination of: – The shifted value A[3:2] – The current value of the DP SELECT register
1000	ABORT [35 bits]	*Abort Register* – Bits 31:1 = Reserved – Bit 0 = DAPABORT: write 1 to generate a DAP abort.

Table 162. 32-bit debug port registers addressed through the shifted value A[3:2]

Address	A(3:2) value	Description
0x0	00	Reserved
0x4	01	DP CTRL/STAT register. Used to: – Request a system or debug power-up – Configure the transfer operation for AP accesses – Control the pushed compare and pushed verify operations. – Read some status flags (overrun, power-up acknowledges)
0x8	10	DP SELECT register: Used to select the current access port and the active 4-words register window. – Bits 31:24: APSEL: select the current AP – Bits 23:8: reserved – Bits 7:4: APBANKSEL: select the active 4-words register window on the current AP – Bits 3:0: reserved
0xC	11	DP RDBUFF register: Used to allow the debugger to get the final result after a sequence of operations (without requesting new JTAG-DP operation)

26.8 SW debug port

26.8.1 SW protocol introduction

This synchronous serial protocol uses two pins:

- SWCLK: clock from host to target
- SWDIO: bidirectional

The protocol allows two banks of registers (DPACC registers and APACC registers) to be read and written to.

Bits are transferred LSB-first on the wire.

For SWDIO bidirectional management, the line must be pulled-up on the board (100 KΩ recommended by ARM).

Each time the direction of SWDIO changes in the protocol, a turnaround time is inserted where the line is not driven by the host nor the target. By default, this turnaround time is one bit time, however this can be adjusted by configuring the SWCLK frequency.

26.8.2 SW protocol sequence

Each sequence consist of three phases:

1. Packet request (8 bits) transmitted by the host
2. Acknowledge response (3 bits) transmitted by the target
3. Data transfer phase (33 bits) transmitted by the host or the target

Table 163. Packet request (8-bits)

Bit	Name	Description
0	Start	Must be “1”
1	APnDP	0: DP Access 1: AP Access
2	RnW	0: Write Request 1: Read Request
4:3	A(3:2)	Address field of the DP or AP registers (refer to *Table 162*)
5	Parity	Single bit parity of preceding bits
6	Stop	0
7	Park	Not driven by the host. Must be read as “1” by the target because of the pull-up

Refer to the *Cortex-M3 r1p1 TRM* for a detailed description of DPACC and APACC registers.

The packet request is always followed by the turnaround time (default 1 bit) where neither the host nor target drive the line.

Table 164. ACK response (3 bits)

Bit	Name	Description
0..2	ACK	001: FAULT 010: WAIT 100: OK

The ACK Response must be followed by a turnaround time only if it is a READ transaction or if a WAIT or FAULT acknowledge has been received.

Table 165. DATA transfer (33 bits)

Bit	Name	Description
0..31	WDATA or RDATA	Write or Read data
32	Parity	Single parity of the 32 data bits

The DATA transfer must be followed by a turnaround time only if it is a READ transaction.

26.8.3 SW-DP state machine (Reset, idle states, ID code)

The State Machine of the SW-DP has an internal ID code which identifies the SW-DP. It follows the JEP-106 standard. This ID code is the default ARM one and is set at **0x1BA01477** (corresponding to Cortex-M3 r1p1).

Note: *Note that the SW-DP state machine is inactive until the target reads this ID code.*

- The SW-DP state machine is in RESET STATE either after power-on reset, or after the DP has switched from JTAG to SWD or after the line is high for more than 50 cycles
- The SW-DP state machine is in IDLE STATE if the line is low for at least two cycles after RESET state.
- After RESET state, it is **mandatory** to first enter into an IDLE state AND to perform a READ access of the DP-SW ID CODE register. Otherwise, the target will issue a FAULT acknowledge response on another transactions.

Further details of the SW-DP state machine can be found in the *Cortex-M3 r1p1 TRM* and the *CoreSight Design Kit r1p0 TRM*.

26.8.4 DP and AP read/write accesses

- Read accesses to the DP are not posted: the target response can be immediate (if ACK=OK) or can be delayed (if ACK=WAIT).
- Read accesses to the AP are posted. This means that the result of the access is returned on the next transfer. If the next access to be done is NOT an AP access, then the DP-RDBUFF register must be read to obtain the result.
  The READOK flag of the DP-CTRL/STAT register is updated on every AP read access or RDBUFF read request to know if the AP read access was successful.
- The SW-DP implements a write buffer (for both DP or AP writes), that enables it to accept a write operation even when other transactions are still outstanding. If the write buffer is full, the target acknowledge response is “WAIT”. With the exception of

IDCODE read or CTRL/STAT read or ABORT write which are accepted even if the write buffer is full.

- Because of the asynchronous clock domains SWCLK and HCLK, two extra SWCLK cycles are needed after a write transaction (after the parity bit) to make the write effective internally. These cycles should be applied while driving the line low (IDLE state)
  This is particularly important when writing the CTRL/STAT for a power-up request. If the next transaction (requiring a power-up) occurs immediately, it will fail.

26.8.5 SW-DP registers

Access to these registers are initiated when APnDP=0

Table 166. SW-DP registers

A(3:2)	R/W	CTRLSEL bit of SELECT register	Register	Notes
00	Read		IDCODE	The manufacturer code is not set to ST code. 0x1BA01477 (identifies the SW-DP)
00	Write		ABORT	
01	Read/Write	0	DP-CTRL/STAT	Purpose is to: – request a system or debug power-up – configure the transfer operation for AP accesses – control the pushed compare and pushed verify operations. – read some status flags (overrun, power-up acknowledges)
01	Read/Write	1	WIRE CONTROL	Purpose is to configure the physical serial port protocol (like the duration of the turnaround time)
10	Read		READ RESEND	Enables recovery of the read data from a corrupted debugger transfer, without repeating the original AP transfer.
10	Write		SELECT	The purpose is to select the current access port and the active 4-words register window
11	Read/Write		READ BUFFER	This read buffer is useful because AP accesses are posted (the result of a read AP request is available on the next AP transaction), This read buffer captures data from the AP, presented as the result of a previous read, without initiating a new transaction

26.8.6 SW-AP registers

Access to these registers are initiated when APnDP=1

There are many AP Registers (see AHB-AP) addressed as the combination of:

- The shifted value A[3:2]
- The current value of the DP SELECT register

26.9 AHB-AP (AHB Access Port) - valid for both JTAG-DP or SW-DP

Features:

- System access is independent of the processor status.
- Either SW-DP or JTAG-DP accesses AHB-AP.
- The AHB-AP is an AHB master into the Bus Matrix. Consequently, it can access all the data buses (Dcode Bus, System Bus, internal and external PPB bus) but the ICode bus.
- Bitband transactions are supported.
- AHB-AP transactions bypass the FPB.

The address of the 32-bits AHP-AP resisters are 6-bits wide (up to 64 words or 256 bytes) and consists of:

f) Bits [8:4] = the bits[7:4] APBANKSEL of the DP SELECT register

g) Bits [3:2] = the 2 address bits of A(3:2) of the 35-bit packet request for SW-DP.

The AHB-AP of the Cortex-M3 includes 9 x 32-bits registers:

Table 167. Cortex-M3 AHB-AP registers

Address offset	Register name	Notes
0x00	AHB-AP Control and Status Word	Configures and controls transfers through the AHB interface (size, hprot, status on current transfer, address increment type
0x04	AHB-AP Transfer Address	
0x0C	AHB-AP Data Read/Write	
0x10	AHB-AP Banked Data 0	Directly maps the 4 aligned data words without rewriting the Transfer Address Register.
0x14	AHB-AP Banked Data 1	
0x18	AHB-AP Banked Data 2	
0x1C	AHB-AP Banked Data 3	
0xF8	AHB-AP Debug ROM Address	Base Address of the debug interface
0xFC	AHB-AP ID Register	

Refer to the *Cortex-M3 r1p1 TRM* for further details.

26.10 Core debug

Core debug is accessed through the core debug registers. Debug access to these registers is by means of the *Advanced High-performance Bus* (AHB-AP) port. The processor can access these registers directly over the internal *Private Peripheral Bus* (PPB).

It consists of 4 registers:

Table 168. Core debug registers

Register	Description
DHCSR	*The 32-bit Debug Halting Control and Status Register* This provides status information about the state of the processor enable core debug halt and step the processor
DCRSR	*The 17-bit Debug Core Register Selector Register:* This selects the processor register to transfer data to or from.
DCRDR	*The 32-bit Debug Core Register Data Register:* This holds data for reading and writing registers to and from the processor selected by the DCRSR (Selector) register.
DEMCR	*The 32-bit Debug Exception and Monitor Control Register:* This provides Vector Catching and Debug Monitor Control. This register contains a bit named ***TRCENA*** which enable the use of a TRACE.

Note: ***Important****: these registers are not reset by a system reset. They are only reset by a power-on reset.*

Refer to the *Cortex-M3 r1p1 TRM* for further details.

To Halt on reset, it is necessary to:

- enable the bit0 (VC_CORRESET) of the Debug and Exception Monitor Control Register
- enable the bit0 (C_DEBUGEN) of the Debug Halting Control and Status Register.

26.11 Capability of the debugger host to connect under system reset

The STM32F10xxx MCU reset system comprises the following reset sources:

- POR (Power On Reset) which asserts a RESET at each power-up.
- Internal watchdog reset
- Software reset
- External reset

The Cortex-M3 differentiates the reset of the debug part (generally PORRESETn) and the other one (SYSRESETn)

This way, it is possible for the debugger to connect under System Reset, programming the Core Debug Registers to halt the core when fetching the reset vector. Then the host can release the system reset and the core will immediately halt without having executed any instructions. In addition, it is possible to program any debug features under System Reset.

Note: *It is highly recommended for the debugger host to connect (set a breakpoint in the reset vector) under system reset.*

26.12 FPB (Flash patch breakpoint)

The FPB unit:

- implements hardware breakpoints
- patches code and data from code space to system space. This feature gives the possibility to correct software bugs located in the Code Memory Space.

The use of a Software Patch or a Hardware Breakpoint is exclusive.

The FPB consists of:

- 2 literal comparators for matching against literal loads from Code Space and remapping to a corresponding area in the System Space.
- 6 instruction comparators for matching against instruction fetches from Code Space. They can be used either to remap to a corresponding area in the System Space or to generate a Breakpoint Instruction to the core.

26.13 DWT (data watchpoint trigger)

The DWT unit consists of four comparators. They are configurable as:

- a hardware watchpoint or
- a trigger to an ETM or
- a PC sampler or
- a data address sampler.

The DWT also provides some means to give some profiling informations. For this, some counters are accessible to give the number of:

- Clock cycle
- Folded instructions
- Load store unit (LSU) operations
- Sleep cycles
- CPI (clock per instructions)
- Interrupt overhead

26.14 ITM (instrumentation trace macrocell)

26.14.1 General description

The ITM is an application-driven trace source that supports *printf* style debugging to trace *Operating System* (OS) and application events, and emits diagnostic system information. The ITM emits trace information as packets which can be generated as:

- **Software trace.** Software can write directly to the ITM stimulus registers to emit packets.
- **Hardware trace.** The DWT generates these packets, and the ITM emits them.
- **Time stamping.** Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp. The Cortex-M3 clock or the bit clock rate of the *Serial Wire Viewer* (SWV) output clocks the counter.

The packets emitted by the ITM are output to the TPIU (Trace Port Interface Unit). The formatter of the TPIU adds some extra packets (refer to TPIU) and then output the complete packets sequence to the debugger host.

The bit TRCEN of the Debug Exception and Monitor Control Register must be enabled before you program or use the ITM.

26.14.2 Timestamp packets, synchronization and overflow packets

Timestamp packets encode timestamp information, generic control and synchronization. It uses a 21-bit timestamp counter (with possible prescalers) which is reset at each time stamp packet emission. This counter can be either clocked by the CPU clock or the SWV clock.

A synchronization packet consists of 6 bytes equal to 0x80_00_00_00_00_00 which is emitted to the TPIU as 00 00 00 00 00 80 (LSB emitted first).

A synchronization packet is a timestamp packet control. It is emitted at each DWT trigger.

For this, the DWT must be configured to trigger the ITM: the bit CYCCNTENA (bit0) of the DWT Control Register must be set. In addition, the bit2 (SYNCENA) of the ITM Trace Control Register must be set.

Note: If the SYNENA bit is not set, the DWT generates Synchronization triggers to the TPIU which will send only TPIU synchronization packets and not ITM synchronization packets.

An overflow packet consists is a special timestamp packets which indicates that data has been written but the FIFO was full.

Table 169. Main ITM registers

Address	Register	Details
@E0000FB0	ITM Lock Access	Write 0xC5ACCE55 to unlock Write Access to the other ITM registers
@E0000E80	ITM Trace Control	Bits 31-24 = Always 0
		Bits 23 = Busy
		Bits 22-16 = 7-bits ATB ID which identifies the source of the trace data.
		Bits 15-10 = Always 0
		Bits 9:8 = TSPrescale = Time Stamp Prescaler
		Bits 7-5 = Reserved
		Bit 4 = SWOENA = Enable SWV behavior (to clock the timestamp counter by the SWV clock).
		Bit 3 = DWTENA: Enable the DWT Stimulus
		Bit 2 = SYNCENA: this bit must be to 1 to enable the DWT to generate synchronization triggers so that the TPIU can then emit the synchronization packets.
		Bit 1 = TSENA (Timestamp Enable)
		Bit 0 = ITMENA: Global Enable Bit of the ITM

Table 169. Main ITM registers

Address	Register	Details
@E0000E40	ITM Trace Privilege	Bit 3: mask to enable tracing ports31:24
		Bit 2: mask to enable tracing ports23:16
		Bit 1: mask to enable tracing ports15:8
		Bit 0: mask to enable tracing ports7:0
@E0000E00	ITM Trace Enable	Each bit enables the corresponding Stimulus port to generate trace.
@E0000000-E000007C	Stimulus Port Registers 0-31	Write the 32-bits data on the selected Stimulus Port (32 available) to be traced out.

Example of configuration

To output a simple value to the TPIU:

- Configure the TPIU and assign TRACE I/Os by configuring the DBGMCU_CR (refer to *Section 26.16.2: TRACE pin assignment* and *Section 26.15.3: Debug MCU configuration register*)
- Write 0xC5ACCE55 to the ITM Lock Access Register to unlock the write access to the ITM registers
- Write 0x00010005 to the ITM Trace Control Register to enable the ITM with Sync enabled and an ATB ID different from 0x00
- Write 0x1 to the ITM Trace Enable Register to enable the Stimulus Port 0
- Write 0x1 to the ITM Trace Privilege Register to unmask stimulus ports 7:0
- Write the value to output in the Stimulus Port Register 0: this can be done by software (using a printf function)

26.15 MCU debug component (MCUDBG)

The MCU debug component helps the debugger provide support for:

- Low-power modes
- Clock control for timers, watchdog and bxCAN during a breakpoint
- Control of the trace pins assignment

26.15.1 Debug support for low-power modes

To enter low-power mode, the instruction WFI or WFE must be executed.

The MCU implements several low-power modes which can either deactivate the CPU clock or reduce the power of the CPU.

The core does not allow FCLK or HCLK to be turned off during a debug session. As these are required for the debugger connection, during a debug, they must remain active. The MCU integrates special means to allow the user to debug software in low-power modes.

For this, the debugger host must first set some debug configuration registers to change the low-power mode behavior:

- In Sleep mode, DBG_SLEEP bit of DBGMCU_CR register must be previously set by the debugger. This will feed HCLK with the same clock that is provided to FCLK (system clock previously configured by the software).
- In STOP mode, the bit DBG_STOP must be previously set by the debugger. This will enable the internal RC oscillator clock to feed FCLK and HCLK in STOP mode.

26.15.2 Debug support for timers, watchdog, bxCAN and I^{2}C

During a breakpoint, it is necessary to choose how the counter of timers and watchdog should behave:

- they can continue to count inside a breakpoint. This is usually required when a PWM is controlling a motor, for example.
- they can stop to count inside a breakpoint. This is required for watchdog purposes.

For the bxCAN, the user can choose to block the update of the receive register during a breakpoint.

For the I^{2}C, the user can choose to block the SMBUS timeout during a breakpoint.

26.15.3 Debug MCU configuration register

This register allows the configuration of the MCU under DEBUG. This concerns:

- Low-power mode support
- Timer and watchdog counters support
- bxCAN communication support
- Trace pin assignment

This DBGMCU_CR is mapped on the External PPB bus at address 0xE004 2004

It is asynchronously reset by the PORESET (and not the system reset). It can be written by the debugger under system reset.

If the debugger host does not support these features, it is still possible for the user software to write to these registers.

DBGMCU_CR

Address: 0xE0042004

Only 32-bit access supported

POR Reset: 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved											DBG_ TIM7_ STOP	DBG_ TIM6_ STOP	DBG_ TIM5_ STOP	DBG_ TIM8_ STOP	DBG_I2C2 _SMBUS_ TIMEOUT
											rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBG_I2C1 _SMBUS_ TIMEOUT	DBG_ CAN_ STOP	DBG_ TIM4_ STOP	DBG_ TIM3_ STOP	DBG_ TIM2_ STOP	DBG_ TIM1_ STOP	DBG_ WWDG_ STOP	DBG_ IWDG STOP	TRACE_ MODE [1:0]		TRACE_ IOEN	Reserved		DBG_ STAND BY	DBG_ STOP	DBG_ SLEEP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw

Bits 31:21 Reserved, must be kept cleared.

Bits 20:17 **DBG_TIMx_STOP:** TIMx counter stopped when core is halted (x=8..5)

0: The clock of the involved timer counter is fed even if the core is halted, and the outputs behave normally.

1: The clock of the involved timer counter is stopped when the core is halted, and the outputs are disabled (as if there were an emergency stop in response to a break event).

Bit 16 **DBG_I2C2_SMBUS_TIMEOUT:** SMBUS timeout mode stopped when Core is halted

0: Same behavior as in normal mode.

1: The SMBUS timeout is frozen

Bit 15 **DBG_I2C1_SMBUS_TIMEOUT:** SMBUS timeout mode stopped when Core is halted

0: Same behavior as in normal mode.

1: The SMBUS timeout is frozen.

Bit 14 **DBG_CAN_STOP:** Debug CAN stopped when Core is halted

0: Same behavior as in normal mode.

1: The CAN receive registers are frozen.

Bits 13:10 **DBG_TIMx_STOP:** TIMx counter stopped when core is halted (x=4..1)

0: The clock of the involved Timer Counter is fed even if the core is halted.

1: The clock of the involved Timer counter is stopped when the core is halted.

Bit 9 **DBG_WWDG_STOP:** Debug window watchdog stopped when core is halted

0: The window watchdog counter clock continues even if the core is halted.

1: The window watchdog counter clock is stopped when the core is halted.

Bit 8 **DBG_IWDG_STOP:** Debug independent watchdog stopped when core is halted

0: The watchdog counter clock continues even if the core is halted.

1: The watchdog counter clock is stopped when the core is halted.

Bits 7:5 **TRACE_MODE[1:0] and TRACE_IOEN**: Trace pin assignment control

– *With TRACE_IOEN=0:*

TRACE_MODE=xx: TRACE pins not assigned (default state)

– *With TRACE_IOEN=1:*

TRACE_MODE=00: TRACE pin assignment for Asynchronous Mode

TRACE_MODE=01: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1

TRACE_MODE=10: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2

TRACE_MODE=11: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

Bit 4:3 Reserved, must be kept cleared.

Bit 2 **DBG_STANDBY:** Debug Standby mode

0: (FCLK=Off, HCLK=Off) The whole digital part is unpowered.

From software point of view, exiting from Standby is identical than fetching reset vector (except a few status bit indicated that the MCU is resuming from Standby)

1: (FCLK=On, HCLK=On) In this case, the digital part is not unpowered and FCLK and HCLK are provided by the internal RC oscillator which remains active. In addition, the MCU generate a system reset during Standby mode so that exiting from Standby is identical than fetching from reset

Bit 1 **DBG_STOP:** Debug Stop mode

0: (FCLK=Off, HCLK=Off) In STOP mode, the clock controller disables all clocks (including HCLK and FCLK). When exiting from STOP mode, the clock configuration is identical to the one after RESET (CPU clocked by the 8 MHz internal RC oscillator (HSI)). Consequently, the software must reprogram the clock controller to enable the PLL, the Xtal, etc.

1: (FCLK=On, HCLK=On) In this case, when entering STOP mode, FCLK and HCLK are provided by the internal RC oscillator which remains active in STOP mode. When exiting STOP mode, the software must reprogram the clock controller to enable the PLL, the Xtal, etc. (in the same way it would do in case of DBG_STOP=0)

Bit 0 **DBG_SLEEP:** Debug Sleep mode

0: (FCLK=On, HCLK=Off) In Sleep mode, FCLK is clocked by the system clock as previously configured by the software while HCLK is disabled.

In Sleep mode, the clock controller configuration is not reset and remains in the previously programmed state. Consequently, when exiting from Sleep mode, the software does not need to reconfigure the clock controller.

1: (FCLK=On, HCLK=On) In this case, when entering Sleep mode, HCLK is fed by the same clock that is provided to FCLK (system clock as previously configured by the software).

26.16 TPIU (trace port interface unit)

26.16.1 Introduction

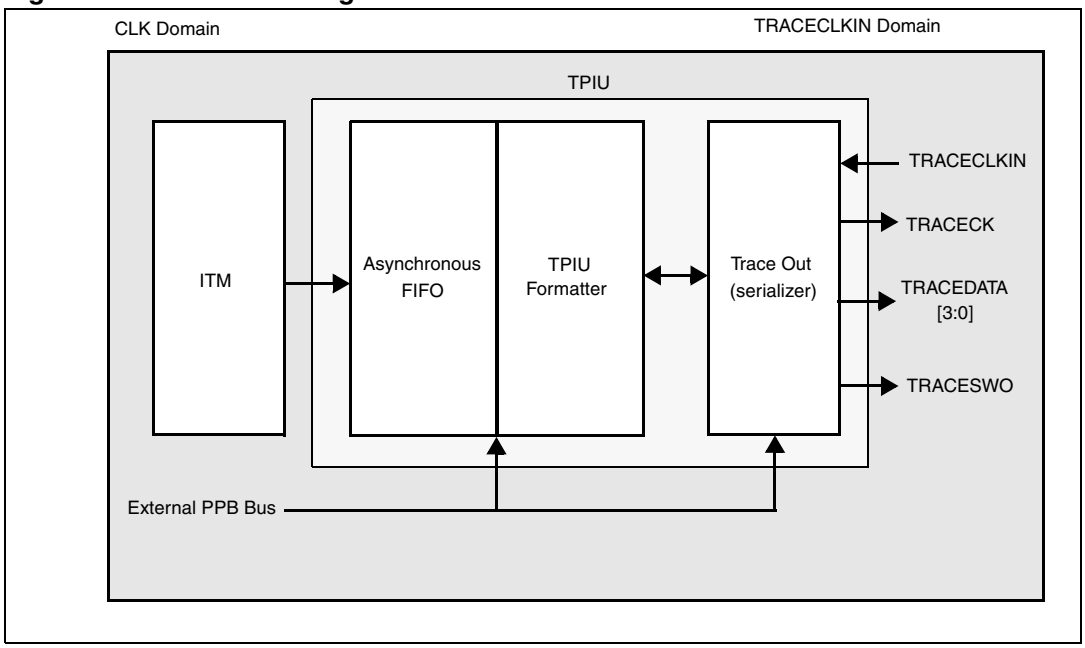
The TPIU acts as a bridge between the on-chip trace data from the ITM.

The output data stream encapsulates the trace source ID, that is then captured by a *Trace Port Analyzer* (TPA).

The core embeds a simple TPIU, especially designed for low-cost debug (consisting of a special version of the CoreSight TPIU).

The TPIU only supports ITM debug trace which is a *limited trace* as it only outputs information coming from the ITM.

Figure 261. TPIU block diagram



26.16.2 TRACE pin assignment

- Asynchronous mode

  The asynchronous mode requires 1 extra pin and is available on all packages. It is only available if using Serial Wire mode (not in JTAG mode).

Table 170. Asynchronous TRACE pin assignment

TPUI pin name	Trace synchronous mode		STM32F10xxx pin assignment
	Type	Description	
TRACESWO	O	TRACE Async Data Output	PB3

- Synchronous mode

  The synchronous mode requires from 2 to 6 extra pins depending on the data trace size and is only available in the larger packages. In addition it is available in JTAG mode and in Serial Wire mode and provides better bandwidth output capabilities than asynchronous trace.

Table 171. Synchronous TRACE pin assignment

TPUI pin name	Trace synchronous mode		STM32F10xxx pin assignment
	Type	Description	
TRACECK	O	TRACE Clock	PE2
TRACED[3:0]	O	TRACE Sync Data Outputs Can be 1, 2 or 4.	PE[6:3]

TPUI TRACE pin assignment

By default, these pins are NOT assigned. They can be assigned by setting the IOTRACEN and IOTRACEMODE bits of the ***MCU Debug Component Configuration Register***. This configuration has to be done by the debugger host.

In addition, the number of pins to assign depends on the trace configuration (asynchronous or synchronous).

- **Asynchronous mode**: 1 extra pin is needed
- **Synchronous mode**: from 2 to 5 extra pins are needed depending on the size of the data trace port register (1, 2 or 4):
  - TRACECK
  - TRACED(0) if port size is configured to 1, 2 or 4
  - TRACED(1) if port size is configured to 2 or 4
  - TRACED(2) if port size is configured to 4
  - TRACED(3) if port size is configured to 4

To assign the TRACE pin, the debugger host must program the bits TRACE_IOEN and TRACE_MODE[1:0] of the Debug MCU configuration Register (DBGMCU_CR). By default the TRACE pins are not assigned.

This register is mapped on the external PPB and is reset by the PORESET (and not by the SYSTEM reset). It can be written by the debugger under SYSTEM reset.

Table 172. Flexible TRACE pin assignment

DBGMCU_CR register		Pins assigned for:	TRACE I/O pin assigned					
TRACE_IOEN	**TRACE_MODE[1:0]**		**PB3 /** JTDO/ TRACESWO	**PE2 /** TRACECK	**PE3 /** TRACED[0]	**PE4 /** TRACED[1]	**PE5 /** TRACED[2]	**PE6 /** TRACED[3]
0	XX	No Trace (default state)	Released (1)	Released (usable as GPIO)				
1	00	Asynchronous Trace	TRACESWO					
1	01	Synchronous Trace 1 bit	Released (1)	TRACECK	TRACED[0]			
1	10	Synchronous Trace 2 bit		TRACECK	TRACED[0]	TRACED[1]		
1	11	Synchronous Trace 4 bit		TRACECK	TRACED[0]	TRACED[1]	TRACED[2]	TRACED[3]

(1) When Serial Wire mode is used, it is released. But when JTAG is used, it is assigned to JTDO.

Note: By default, the TRACECLKIN input clock of the TPIU is tied to GND. It is assigned to HCLK two clock cycles after the bit TRACE_IOEN has been set.

The debugger must then program the Trace Mode by writing the PROTOCOL[1:0] bits in the SPP_R (Selected Pin Protocol) register of the TPIU.

- PROTOCOL=00: Trace Port Mode (synchronous)
- PROTOCOL=01 or 10: Serial Wire (Manchester or NRZ) Mode (asynchronous mode). Default state is 01

It then also configures the TRACE port size by writing the bits [3:0] in the CPSPS_R (Current Sync Port Size Register) of the TPIU:

- 0x1 for 1 pin (default state)
- 0x2 for 2 pins
- 0x8 for 4 pins

26.16.3 TPUI formatter

The formatter protocol outputs data in 16-byte frames:

- seven bytes of data
- eight bytes of mixed-use bytes consisting of:
  - 1 bit (LSB) to indicate it is a DATA byte (‘0’) or an ID byte (‘1’).
  - 7 bits (MSB) which can be data or change of source ID trace.
- one byte of auxiliary bits where each bit corresponds to one of the eight mixed-use bytes:
  - if the corresponding byte was a data, this bit gives bit0 of the data.
  - if the corresponding byte was an ID change, this bit indicates when that ID change takes effect.

Note: Refer to the ARM CoreSight Architecture Specification v1.0 (ARM IHI 0029B) for further information

Use of the formatter for STM32F10xxx MCU

For STM32F10xxx MCU, there is only one TRACE source (the ITM). But the formatter can not be disabled and must be used in bypass mode because the TRACECTL pin is not assigned. This way, the Trace Port Analyzer can decode part of the formatter protocol to determine the position of the trigger.

26.16.4 TPUI frame synchronization packets

The TPUI can generate two types of synchronization packets:

- The Frame Synchronization packet (or Full Word Synchronization packet)

  It consists of the word: 0x7F_FF_FF_FF (LSB emitted first). This sequence can not occur at any other time provided that the ID source code 0x7F has not been used.

  It is output periodically ***between*** frames.

  In continuous mode, the TPA must discard all these frames once a synchronization frame has been found.
- The Half-Word Synchronization packet

  It consists of the half word: 0x7F_FF (LSB emitted first).

  It is output periodically ***between or within*** frames.

  These packets are only generated in continuous mode and enable the TPA to detect that the TRACE port is in IDLE mode (no TRACE to be captured). When detected by the TPA, it must be discarded.

26.16.5 Emission of synchronization frame packet

There is no Synchronization Counter register implemented in the TPIU of the core. Consequently, the synchronization trigger can only be generated by the **DWT**. Refer to the registers DWT Control Register (bits SYNCTAP[11:10]) and the DWT Current PC Sampler Cycle Count Register.

The TPUI Frame synchronization packet (0x7F_FF_FF_FF) is emitted:

- after each TPIU reset release. This reset is synchronously released with the rising edge of TRACECLKIN clock. This means that this packet is emitted once the bit IO_TRACEN of the DBGMCU_CFG register has been set. In this case, the word 0x7F_FF_FF_FF is not followed by any formatted packet.
- at each DWT trigger (assuming DWT has been previously configured). Two cases occur:
  - If the bit SYNENA of the ITM is reset, only the word 0x7F_FF_FF_FF is emitted without any formatted stream which follows.
  - If the bit SYNENA of the ITM is set, then the ITM synchronization packets will follow (0x80_00_00_00_00_00), formatted by the TPUI (trace source ID added).

26.16.6 Synchronous mode

The trace data output size can be configured to 4, 2 or 1 pin: TRACED(3:0)

The output clock is output to the debugger (TRACECK)

Here, TRACECLKIN is driven internally and is connected to HCLK only when TRACE is used.

Note: In this synchronous mode, it is not required to provide a stable clock frequency.

The TRACE I/Os (including TRACECK) are driven by the rising edge of TRACLKIN (equal to HCLK). Consequently, the output frequency of TRACECK is equal to HCLK/2.

26.16.7 Asynchronous mode

This is a low cost alternative to output the trace using only 1 pin: this is the asynchronous output pin TRACESWO. Obviously there is a limited bandwidth.

TRACESWO is multiplexed with JTDO when using the SW-DP pin. This way, this functionality is available in all STM32F10xxx packages.

This asynchronous mode requires a constant frequency for TRACECLKIN. For the standard UART (NRZ) capture mechanism, 5% accuracy is needed. The Manchester encoded version is tolerant up to 10%.

26.16.8 TRACECLKIN connection inside STM32F10xxx

In STM32F10xxx, this TRACECLKIN input is internally connected to HCLK. This means that when in asynchronous trace mode, the application is restricted to use to time frames where the CPU frequency is stable.

Note: ***Important:*** *when using asynchronous trace: it is important to be aware that:*

The default clock of the STM32F10xxx MCU is the internal RC oscillator. Its frequency under reset is different from the one after reset release. This is because the RC calibration is the default one under system reset and is updated at each system reset release.

Consequently, the Trace Port Analyzer (TPA) should not enable the trace (with the bit IOTRACEN) under system reset, because a Synchronization Frame Packet will be issued with a different bit time than trace packets which will be transmitted after reset release.

26.16.9 TPIU registers

The TPIU APB registers can be read and written only if the bit TRCENA of the Debug Exception and Monitor Control Register (DEMCR) is set. Otherwise, the registers are read as zero (the output of this bit enables the PCLK of the TPIU).

Table 173. Important TPIU registers

Address	Register	Description
0xE0040004	Current port size	Allows the trace port size to be selected: Bit 0: Port size = 1 Bit 1: Port size = 2 Bit 2: Port size = 3, not supported Bit 3: Port Size = 4 Only 1 bit must be set. By default, the port size is one bit. (0x00000001)
0xE00400F0	Selected pin protocol	Allows the Trace Port Protocol to be selected: Bit1:0= 00: Sync Trace Port Mode 01: Serial Wire Output - manchester (default value) 10: Serial Wire Output - NRZ 11: reserved

Table 173. Important TPIU registers (continued)

Address	Register	Description
0xE0040304	Formatter and flush control	Bit 31-9 = always ‘0’ Bit 8 = TrigIn = always ‘1’ to indicate that triggers are indicated Bit 7-4 = always 0 Bit 3-2 = always 0 Bit 1 = EnFCont. In Sync Trace mode (Select_Pin_Protocol register bit1:0=00), this bit is forced to ‘1’: the formatter is automatically enabled in continuous mode. In asynchronous mode (Select_Pin_Protocol register bit1:0 <> 00), this bit can be written to activate or not the formatter. Bit 0 = always 0 The resulting default value is 0x102 **Note:** In synchronous mode, because the TRACECTL pin is not mapped outside the chip, the formatter is always enabled in continuous mode -this way the formatter inserts some control packets to identify the source of the trace packets).
0xE0040300	Formatter and flush status	Not used in Cortex-M3, always read as 0x00000008

26.16.10 Example of configuration

- Set the bit TRCENA in the Debug Exception and Monitor Control Register (DEMCR)
- Write the TPIU Current Port Size Register to the desired value (default is 0x1 for a 1-bit port size)
- Write TPIU Formatter and Flush Control Register to 0x102 (default value)
- Write the TPIU Select Pin Protocol to select the sync or async mode. Example: 0x2 for async NRZ mode (UART like)
- Write the DBGMCU Control Register to 0x20 (bit IO_TRACEN) to assign TRACE I/Os for async mode. A TPIU Sync packet is emitted at this time (FF_FF_FF_7F)
- Configure the ITM and write the ITM Stimulus register to output a value

26.17 DBG register map

The following table summarizes the Debug registers.

Table 174. DBG register map and reset values

Addr.	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xE0042000	**DBGMCU_ IDCODE**	REV_ID																Reserved				DEV_ID											
	Reset value[1]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					X	X	X	X	X	X	X	X	X	X	X	X
0xE0042004	**DBGMCU_CR**	Reserved											DBG_TIM7_STOP	DBG_TIM6_STOP	DBG_TIM5_STOP	DBG_TIM8_STOP	DBG_I2C2_SMBUS_TIMEOUT	DBG_I2C1_SMBUS_TIMEOUT	DBG_CAN_STOP	DBG_TIM4_STOP	DBG_TIM3_STOP	DBG_TIM2_STOP	DBG_TIM1_STOP	DBG_WWDG_STOP	DBG_IWDG_STOP	TRACE_MODE[1:0]		TRACE_IOEN	Reserved			DBG_STANDBY	DBG_STOP	DBG_Sleep
	Reset value												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			0	0	0

1. The reset value is product dependent. For more information, refer to *Section 26.6.1: MCU device ID code*.

27 Revision history

Table 175. Document revision history

Date	Revision	Changes
19-Oct-2007	1	Document reference number changed from UM0306 to RM008. The changes below were made with reference to revision 1 of 01-Jun-2007 of UM0306. EXTSEL[2:0] and JEXTSEL[2:0] removed from *Table 43: ADC pins on page 146* and V_{REF+} range modified in Remarks column. Notes added to *Section 10.3.9 on page 149*, *Section 10.9.2 on page 157*, *Section 10.9.7 on page 160* and *Section 10.9.9 on page 161*. SPI_CR2 corrected to SPI_CR1 in *1 clock and 1 bidirectional data wire on page 539*. f_{CPU} frequency changed to f_{PCLK} in *Section 22.2: SPI and I2S main features on page 532*. *Section 22.3.6: CRC calculation on page 540* and *Section 22.3.7: SPI communication using DMA (direct memory addressing) on page 541* modified. Note added to bit 13 description changed in *Section 22.5.1: SPI Control Register 1 (SPI_CR1) (not used in I2S mode) on page 557*. Note for bit 4 modified in *Section 22.5.3: SPI status register (SPI_SR) on page 561*. *On 64-pin packages on page 46* modified. *Section 7.3.2: Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1 on page 105* updated. Description of SRAM at address 0x4000 6000 modified in *Figure 2: Memory map on page 39* and *Table 1: Register boundary addresses*. Note added to *Section 20.2: USB main features on page 458* and *Section 21.2: bxCAN main features on page 489*. *Figure 3: Power supply overview* and *On 100-pin and 144- pin packages* modified. Formula added to Bits 25:24 description in *CAN bit timing register (CAN_BTR) on page 518*. *Section 9.3: DMA functional description on page 129* modified. *Example of configuration on page 657* modified. MODEx[1:0] bit definitions corrected in *Section 7.2.2: Port configuration register high (GPIOx_CRH) (x=A..G) on page 102*. *Downcounting mode on page 206* modified. *Figure 75: Output stage of capture/compare channel (channel 4) on page 217* and *Figure 77: Output compare mode, toggle on OC1.* modified. OCx output enable conditions modified in *Section 12.3.10: PWM mode on page 221*. *Section 12.3.19: TIMx and external trigger synchronization on page 236* title changed. CC1S, CC2S, CC3S and CC4S definitions modified for (1, 1) bit setting modified in *Section 12.4.7: Capture/compare mode register 1 (TIMx_CCMR1)* and *Section 12.4.8: Capture/compare mode register 2 (TIMx_CCMR2)*. CC1S, CC2S, CC3S and CC4S definitions for (1, 1) bit setting modified in *Section 13.4.7: Capture/compare mode register 1 (TIMx_CCMR1)* and *Section 13.4.8: Capture/compare mode register 2 (TIMx_CCMR2)*. AFIO_EVCR pins modified in *Table 35: AFIO register map and reset values on page 116*. *Section 12.3.6: Input capture mode on page 217* modified.

Table 175. Document revision history (continued)

Date	Revision	Changes
19-Oct-2007 continued	1 continued	*Figure 108: Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6* and *Figure 123: Output compare mode, toggle on OC1.* modified. CKD definition modified in *Section 13.4.1: Control register 1 (TIMx_CR1)*. Bit 8 and Bit 9 added to *Section 5.4.2: RTC clock calibration register (BKP_RTCCR)*. Bit 15 and Bit 16 added to *DBGMCU_CR on page 658*. *Section 23.5: I2C debug mode on page 584* added. Stop and Standby modified in *Table 7: Low-power mode summary*. *Table 9: Sleep-on-exit* modified. *Debug mode on page 54* modified. HSITRIM[4:0] bit description modified in *Section 6.3.1: Clock control register (RCC_CR)*. Note modified in MCO description in *Section 6.3.2: Clock configuration register (RCC_CFGR)*. RCC_CR row modified in *RCC - register map and reset values on page 92*. Bits 15:0 description modified in *Section 7.2.6: Port bit reset register (GPIOx_BRR) (x=A..G)*. *Embedded boot loader on page 41* added. *Figure 9*, *Figure 11*, *Figure 12*, *Figure 13* and *Figure 14* modified. *Section 2.3.3: Embedded Flash memory on page 37* modified. REV_ID bit description added to *DBGMCU_IDCODE on page 646*. Reset value modified in *Clock control register (RCC_CR) on page 74* and HSITRIM[4:0] description modified. *Section 7.1.1 on page 96* modified. Bit definitions modified in *Section 7.2: GPIO registers on page 101*. Wakeup latency description modified in *Table 10: Stop mode*. *Clock control register (RCC_CR)* reset value modified. Note added in ASOS and ASOE bit descriptions in *5.4.2 on page 60*. *Section 26.15.2: Debug support for timers, watchdog, bxCAN and I2C* modified. *Table 174: DBG register map and reset values* updated. *Section 20.5.3: Buffer descriptor table* clarified. *Center-aligned mode (up/down counting) on page 208* and *Center-aligned mode (up/down counting) on page 275* updated. *Figure 79: Center-aligned PWM waveforms (ARR=8) on page 223* and *Figure 125: Center-aligned PWM waveforms (ARR=8) on page 288* modified. RSTCAL description modified in *Section 10.12.3: ADC control register 2 (ADC_CR2)*. Note changed below *Table 64: Watchdog timeout period (with 40 kHz input clock)*. Note added below *Figure 7: Clock tree*. ADC conversion time modified in *Section 10.2: ADC main features*. *Auto-injection on page 149* updated. Note added in *Section 10.9.9: Combined injected simultaneous + interleaved*. Note added to *Section 7.3.2: Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1*. Small text changes. Internal LSI RC frequency changed from 32 to 40 kHz. *Table 64: Watchdog timeout period (with 40 kHz input clock)* updated. Option byte addresses corrected in *Figure 2: Memory map* and *Table 3: Flash module organization (medium-density devices)*. Information block organization modified in *Section 2.3.3: Embedded Flash memory*. External event that trigger ADC conversion is EXTI line instead of external interrupt (see *Section 10: Analog-to-digital converter (ADC)*). *Appendix A: Important notes on page 500* added.

Table 175. Document revision history (continued)

Date	Revision	Changes
20-Nov-2007	2	*Figure 237: USART block diagram* modified. Procedure modified in *Character reception on page 605*. In *Section 24.3.4: Fractional baud rate generation*: – Equation legend modified – *Table 154: Error calculation for programmed baud rates* modified – Note added Small text changes. In *CAN bit timing register (CAN_BTR) on page 518*, bit 15 is reserved. Flash memory organization corrected, *Table 3: Flash module organization (medium-density devices)* modified in *Section 2.3.3: Embedded Flash memory*. Note added below *Figure 3: Power supply overview* in *Section 4.1: Power supplies*. RTCSEL[1:0] bit description modified in *Backup domain control register (RCC_BDCR)*. Names of bits [0:2] corrected for RCC_APB1RSTR and RCC_APB1ENR in *Table 14: RCC - register map and reset values*. Impedance value specified in *A.4: Voltage glitch on ADC input 0 on page 500*. In *Section 22.5.1: SPI Control Register 1 (SPI_CR1) (not used in I2S mode)*, BR[2:0] description corrected. Prescaler buffer behavior specified when an update event occurs (see *upcounting mode on page 270*, *Downcounting mode on page 273* and *Center-aligned mode (up/down counting) on page 275*). AWDCH[4:0] modified in *Section 10.12.2: ADC control register 1 (ADC_CR1)* and bits [26:24] are reserved in *Section 10.12.4: ADC sample time register 1 (ADC_SMPR1)*. CAN_BTR bit 8 is reserved in *Table 146: bxCAN register map and reset values*. *CAN master control register (CAN_MCR) on page 508* corrected. V_{REF+} range corrected in *Table 43: ADC pins* and in *On 100-pin and 144- pin packages on page 46*. *Start condition on page 574* updated. Note removed in *Table 17: CAN1 alternate function remapping*. Note added in *Table 25: Timer 4 alternate function remapping*. In *Section 7.4.2: AF remap and debug I/O configuration register (AFIO_MAPR)*, bit definition modified for USART2_REMAP = 0. In *Section 7.4.3: External interrupt configuration register 1 (AFIO_EXTICR1)*, bit definition modified for SPI1_REMAP = 0. In *Table 173: Important TPIU registers*, at 0xE0040004, bit2 set is not supported. TRACE port size setting corrected in *TPUI TRACE pin assignment on page 662*. *Figure 9*, *Figure 11*, *Figure 12*, *Figure 13* and *Figure 14* modified. *Figure 10: Basic structure of a five-volt tolerant I/O port bit* added. *Table 7.3.1: Using OSC32_IN/OSC32_OUT pins as GPIO ports PC14/PC15 on page 105* added. Bit descriptions modified in *Section 15.4.5* and *Section 15.4.6*. JTAG ID code corrected in *Section 26.6.2: Boundary scan TAP on page 647*. Modified: *Section 17.2: WWDG main features*, *Section 5.2: BKP main features*, *Section 5.3.1: Tamper detection*, *Section 5.3.2: RTC calibration*, *Section 20.3: USB functional description*, *Controlling the downcounter: on page 353*, *Section 4.1.2: Battery backup domain*, *Section 8.2: Introduction*. ASOE bit description modified in *Section 5.4.2: RTC clock calibration register (BKP_RTCCR)*.

Table 175. Document revision history (continued)

Date	Revision	Changes
08-Feb-2008	3	*Figure 3: Power supply overview on page 45* modified. *Section 6.1.2: Power reset on page 67* modified. *Section 6.2: Clocks on page 67* modified. Definition of Bits 26:24 modified in *Section 7.4.2: AF remap and debug I/O configuration register (AFIO_MAPR) on page 111*. AFIO_EVCR bits corrected in *Table 35: AFIO register map and reset values on page 116*. Number of maskable interrupt channels modified in *Section 8.1: Nested vectored interrupt controller (NVIC) on page 117*. *Section 9.3.6: Interrupts on page 133* added. Small text changes. Examples modified in *Figure 85: 6-step generation, COM example (OSSR=1) on page 229*. *Table 56: Output control bits for complementary OCx and OCxN channels with break feature on page 256* modified. Register names modified in *Section 21.6.4: CAN filter registers on page 524*. Small text change in *Section 23.3.3: I2C master mode on page 574*. Bits 5:0 frequency description modified in *Section 23.6.2: Control register 2 (I2C_CR2) on page 587*. *Section 20.3.1: Description of USB blocks on page 460* modified. *Section 22.3.4: Simplex communication on page 539* modified. *Section 22.3.6: CRC calculation on page 540* modified. Note added in *BUSY flag on page 540*. *Section 22.3.9: Disabling the SPI on page 543* added. Appendix A: Important notes, removed.
22-May-2008	4 continued on next page	Reference manual updated to apply to devices containing up to 512 Kbytes of Flash memory (High-density devices). Document restructured. Small text changes. Definitions of Medium-density and High-density devices added to all sections. In *Section 2: Memory and bus architecture on page 33*: – *Figure 1: System architecture on page 33*, *Figure 2: Memory map on page 39*, *Table 1: Register boundary addresses on page 35* updated – Note and text added to *AHB/APB bridges (APB) on page 34* – SRAM size in *Section 2.3.1: Embedded SRAM on page 36* – *Section 2.3.3: Embedded Flash memory on page 37* updated (Flash size, page size, number of pages, *Reading Flash memory*, *Table 4: Flash module organization (high-density devices) on page 39* added) – Prefetch buffer on/off specified in *Reading Flash memory* *bit_number* definition modified in *Section 2.3.2: Bit banding on page 36*. *Section 3: CRC calculation unit on page 42* added (*Table 1: Register boundary addresses on page 35* updated, *Figure 2: Memory map on page 39* updated and CRCEN bit added to *Section 6.3.6: AHB peripheral clock enable register (RCC_AHBENR) on page 84*). *Entering Stop mode on page 51* specified. Updated in *Section 5: Backup registers (BKP) on page 58*: number of backup registers and available storage size and *Section 5.1: BKP introduction*. ASOE definition modified in *Section 5.4.2: RTC clock calibration register (BKP_RTCCR) on page 60*.

Table 175. Document revision history (continued)

Date	Revision	Changes
22-May-2008 continued	4 continued	In *Section 6: Reset and clock control (RCC) on page 66*: – *LSI calibration on page 71* added – *Figure 6: Reset circuit on page 67* updated – *APB2 peripheral reset register (RCC_APB2RSTR) on page 80* updated – *APB1 peripheral reset register (RCC_APB1RSTR) on page 82* updated – *AHB peripheral clock enable register (RCC_AHBENR)* updated – *APB2 peripheral clock enable register (RCC_APB2ENR)* updated – *APB1 peripheral clock enable register (RCC_APB1ENR) on page 87* updated (see *Section Table 14.: RCC - register map and reset values*). – LSERDYIE definition modified in *Clock interrupt register (RCC_CIR)* – HSITRIM[4:0] definition modified in *Clock control register (RCC_CR)* In *Section 7: General-purpose and alternate-function I/Os (GPIOs and AFIOs) on page 94*: – GPIO ports F and G added – In *Section 7.3: Alternate function I/O and debug configuration (AFIO) on page 105* remapping for High-density devices added, note modified under *Section 7.3.2*, *Section 7.3.3 on page 105* modified – *AF remap and debug I/O configuration register (AFIO_MAPR) on page 111* updated Updated in *Section 8: Interrupts and events on page 117*: – number of maskable interrupt channels – number of GPIOs (see *Figure 16: External interrupt/event GPIO mapping*) In *Section 9: DMA controller (DMA) on page 128*: – number of DMA controllers and configurable DMA channels updated – *Figure 17: DMA block diagram on page 129* updated, notes added – Note updated in *Section 9.3.2: Arbiter on page 130* – Note updated in *Section 9.3.6: Interrupts on page 133* – *Figure 18: DMA1 request mapping on page 134* updated – *DMA2 controller on page 135* added In *Section 10: Analog-to-digital converter (ADC) on page 144*: – ADC3 added (*Figure 20: Single ADC block diagram on page 145* updated, *Table 48: External trigger for injected channels for ADC3* added, etc.) *Section 11: Digital-to-analog converter (DAC) on page 178* added. In *Section 12: Advanced-control timers (TIM1&TIM8) on page 199*: – Advanced control timer TIM8 added (see *Figure 46: Advanced-control timer block diagram on page 201*) – TS[2:0] modified in *Section 12.4.3: Slave mode control register (TIMx_SMCR) on page 243*. In *Section 13: General-purpose timer (TIMx) on page 266*: – TIM5 added – *Figure 94: General-purpose timer block diagram on page 268* updated. *Table 59: TIMx Internal trigger connection on page 306* modified. *Section 14: Basic timer (TIM6&7) on page 322* added. RTC clock sources specified in *Section 15.2: RTC main features on page 335*. *Section 15.1: RTC introduction* modified. *Section 18: Flexible static memory controller (FSMC) on page 357* added. *Section 19: SDIO interface (SDIO) on page 403* added.

Table 175. Document revision history (continued)

Date	Revision	Changes
22-May-2008 continued	4 continued	*Figure 204: CAN frames on page 506* modified. Bits 31:21 and bits 20:3 modified in *TX mailbox identifier register (CAN_TIxR) (x=0..2) on page 519*. Bits 31:21 and bits 20:3 modified in *Rx FIFO mailbox identifier register (CAN_RIxR) (x=0..1) on page 522*. *Section 23.3.7: DMA requests on page 581* modified. DMAEN bit 11 description modified in *Section 23.6.2: Control register 2 (I2C_CR2) on page 587*. *Clock phase and clock polarity on page 536* modified. *Transmit sequence on page 538* modified. *Receive sequence on page 539* added. *Reception sequence on page 555* modified. *Underrun flag (UDR) on page 556* modified. I^2S feature added (see *Section 22: Serial peripheral interface (SPI) on page 531*). In *Section 26: Debug support (DBG) on page 640*: – *DBGMCU_IDCODE on page 646* and *DBGMCU_CR on page 658* updated – TMC TAP changed to boundary scan TAP – Address onto which DBGMCU_CR is mapped modified in *Section 26.15.3: Debug MCU configuration register on page 658*. *Section 25: Device electronic signature on page 637* added. REV_ID(15:0) definition modified in *Section 26.6.1: MCU device ID code on page 646*.

Table 175. Document revision history (continued)

Date	Revision	Changes
28-Jul-2008	5	Developed polynomial form updated in *Section 3.2: CRC main features on page 42*. *Figure 3: Power supply overview on page 45* modified. *Section 4.1.2: Battery backup domain on page 46* modified. *Section 6.2.5: LSI clock on page 71* specified. *Section 7.1.4: Alternate functions (AF) on page 97* clarified. Note added to *Table 27: Timer 2 alternate function remapping on page 108*. Bits are write-only in *Section 9.4.2: DMA interrupt flag clear register (DMA_IFCR) on page 138*. Register name modified in *Section 10.3.1: ADC on-off control on page 146*. Recommended sampling time given in *Section 10.10: Temperature sensor on page 162*. Bit attributes modified in *Section 10.12.1: ADC status register (ADC_SR) on page 164*. Note modified for bits 23:0 in *Section 10.12.4: ADC sample time register 1 (ADC_SMPR1) on page 170*. Note added in *Section 11.2: DAC main features on page 178*. Formula updated in *Section 11.3.5: DAC output voltage on page 182*. **DBL[4:0]** description modified in *Section 12.3.19: TIMx and external trigger synchronization on page 236*. *Figure 76 on page 219* and *Figure 122 on page 284* modified. *Section 22.5.3: SPI status register (SPI_SR) on page 561* modified. *Closing the communication on page 576* updated. Notes added to *Section 23.6.8: Clock control register (I2C_CCR) on page 594*. TCK replaced by T_{PCLK1} in *Section 23.6.8* and *Section 23.6.9*. OVR changed to ORE in *Figure 257: USART interrupt mapping diagram on page 624*. *Section 24.6.1: Status register (USART_SR) on page 625* updated. *Slave select (NSS) pin management on page 535* clarified. Small text changes.

Table 175. Document revision history (continued)

Date	Revision	Changes
26-Sep-2008	6	This reference manual also applies to low-density STM32F101xx, STM32F102xx and STM32F103xx devices, and to medium-density STM32F102xx devices. In all sections, definitions of low-density and medium-density devices updated. *Section 1.3: Peripheral availability on page 32* added. *Section 2.3.3: Embedded Flash memory on page 37* updated. *Section 4.1.2: Battery backup domain on page 46* modified. Reset value of *Port input data register (GPIOx_IDR) (x=A..G) on page 102* modified. Note added in *Section 7.4: AFIO registers on page 110*. Note removed from bits 18:0 description in *Section 8.3.6: Pending register (EXTI_PR) on page 126*. *Section 12.2: TIM1&TIM8 main features on page 199* and *Section 13.2: TIMx main features on page 267* updated. In *Section 13.3.15: Timer synchronization on page 296*, TS=000. FSMC_CLK signal direction corrected in *Figure 156: FSMC block diagram on page 358*. “Feedback clock” paragraph removed from *Section 18.5.3: General timing rules on page 366*. In *Section 18.5.6: NOR/PSRAM controller registers on page 385*: reset value modified, WAITEN bit default value after reset is 1, bits [5:6] definition modified, , FACCEN default value after reset specified. NWE signal behavior corrected in *Figure 172: Synchronous multiplexed write mode - PSRAM (CRAM) on page 383*. The FSMC interface does not support COSMO RAM and OneNAND devices, and it does not support the asynchronous wait feature. SRAM and ROM 32 memory data size removed from *Table 75: NOR Flash/PSRAM supported memories and transactions on page 365*. *Data latency versus NOR Flash latency on page 380* modified. Bits 19:16 bits are reserved in *SRAM/NOR-Flash write timing registers 1..4 (FSMC_BWTR1..4) on page 389*. *Section 18.6.3: Timing diagrams for NAND, ATA and PC Card on page 393* modified.Definition of PWID bits modified in *Section 18.6.7: NAND Flash/PC Card controller registers on page 396*. *Section 18.6.6: Error correction code computation ECC (NAND Flash) on page 396* modified. Interrupt Mapper definition modified in *Section 20.3.1: Description of USB blocks on page 460*. USB register and memory base addresses modified in *Section 20.5: USB registers on page 473*. *Section 23.3.8: Packet error checking on page 582* modified. *Section : Start bit detection on page 604* added. PE bit description specified in *Status register (USART_SR) on page 625*. “RAM size register” section removed from *Section 25: Device electronic signature on page 637*. Bit definitions updated in *FIFO status and interrupt register 2..4 (FSMC_SR2..4) on page 398*. Small text changes.

Table 175. Document revision history (continued)

Date	Revision	Changes
23-Dec-2008	7	Memory map figure removed from reference manual. *Section 2.1: System architecture on page 33* modified. *Section 2.4: Boot configuration on page 41* modified. *Exiting Sleep mode on page 50* modified. *Section 5.3.2: RTC calibration on page 59* updated. *Wakeup event management on page 121* updated. *Section 6.3: RCC registers on page 74* updated. *Section 9.2: DMA main features on page 128* updated. *Section 9.3.5: Error management* modified. *Figure 17: DMA block diagram on page 129* modified. *Section 9.3.4: Programmable data width, data alignment and endians on page 131* added. Bit definition modified in *Section 9.4.5: DMA channel x peripheral address register (DMA_CPARx) (x = 1 ..7) on page 141* and *Section 9.4.6: DMA channel x memory address register (DMA_CMARx) (x = 1 ..7) on page 141*. Note added below *Figure 76: PWM input mode timing* and *Figure 122: PWM input mode timing*. FSMC_NWAIT signal direction corrected in *Figure 156: FSMC block diagram on page 358*. Value to set modified for bit 6 in *Table 81: FSMC_BCRx bit fields*, *Table 84: FSMC_BCRx bit fields* and *Table 90: FSMC_BCRx bit fields*. *Table 97: 8-bit NAND Flash*, *Table 98: 16-bit NAND Flash* and *Table 99: 16-bit PC Card* modified. NWAIT and INTR signals separated in *Table 99: 16-bit PC Card*. Note added in PWAITEN bit definition in *PC Card/NAND Flash control registers 2..4 (FSMC_PCR2..4) on page 396*. Bit definitions updated in *FIFO status and interrupt register 2..4 (FSMC_SR2..4) on page 398*. Note modified in ADDHLD and ADDSET bit definitions in *SRAM/NOR-Flash chip-select timing registers 1..4 (FSMC_BTR1..4) on page 387*. Bit 8 is reserved in *PC Card/NAND Flash control registers 2..4 (FSMC_PCR2..4) on page 396*. MEMWAIT[15:8] bit definition modified in *Common memory space timing register 2..4 (FSMC_PMEM2..4) on page 399*. ATTWAIT[15:8] bit definition modified in *Attribute memory space timing registers 2..4 (FSMC_PATT2..4) on page 400*. *Section 18.6.5: NAND Flash pre-wait functionality on page 395* modified. *Figure 173: NAND/PC Card controller timing for common memory access* modified. Note added below *Table 67: NOR/PSRAM bank selection on page 360*. 32-bit external memory access removed from *Table 68: External memory address on page 361* and note added. *Caution:* added to *Section 18.6.1: External memory interface signals on page 391*. NIOS16 description modified in *Table 99: 16-bit PC Card on page 392*. Register description modified in *Attribute memory space timing registers 2..4 (FSMC_PATT2..4) on page 400*. *Resetting the password on page 424* step 2 corrected. write_data signal modified in *Figure 173: NAND/PC Card controller timing for common memory access*. *bxCAN main features on page 489* modified. *Section 23.3.8: Packet error checking on page 582* modified. *Section 26.6.3: Cortex-M3 TAP* modified. DBG_TIMx_STOP positions modified in *DBGMCU_CR on page 658*. Small text changes.

Index

A

ADC_CR1 165
ADC_CR2 167
ADC_DR 175
ADC_HTR 172
ADC_JDRx 175
ADC_JOFRx 171
ADC_JSQR 174
ADC_LTR 172
ADC_SMPR1 170
ADC_SMPR2 171
ADC_SQR1 172
ADC_SQR2 173
ADC_SQR3 174
ADC_SR 164
AFIO_EVCR 110
AFIO_EXTICR1 114
AFIO_EXTICR2 114
AFIO_EXTICR3 115
AFIO_EXTICR4 115
AFIO_MAPR 111

B

BKP_CR 61
BKP_CSR 62
BKP_DRx 60
BKP_RTCCR 60

C

CAN_BTR 518
CAN_ESR 517
CAN_FA1R 526
CAN_FFA1R 526
CAN_FiRx 527
CAN_FM1R 525
CAN_FMR 524
CAN_FS1R 525
CAN_IER 516
CAN_MCR 508
CAN_MSR 510
CAN_RDHxR 524
CAN_RDLxR 523
CAN_RDTxR 522
CAN_RF0R 514
CAN_RF1R 515
CAN_RIxR 522
CAN_TDHxR 521
CAN_TDLxR 521
CAN_TDTxR 520
CAN_TIxR 519
CAN_TSR 511
CRC_DR 43
CRC_IDR 44

D

DBGMCU_CR 658
DBGMCU_IDCODE 646
DMA_CCRx 139
DMA_CMARx 141
DMA_CNDTRx 140
DMA_CPARx 141
DMA_IFCR 138
DMA_ISR 137

E

EXTI_EMR 124
EXTI_FTSR 125
EXTI_IMR 124
EXTI_PR 126
EXTI_RTSR 125
EXTI_SWIER 126

G

GPIOx_BRR 103
GPIOx_BSRR 103
GPIOx_CRH 102
GPIOx_CRL 101
GPIOx_IDR 102
GPIOx_LCKR 104
GPIOx_ODR 103

I

I2C_CCR 594
I2C_CR1 584
I2C_CR2 587
I2C_DR 589
I2C_OAR1 588
I2C_OAR2 589
I2C_SR1 590
I2C_SR2 593
I2C_TRISE 595

IWDG_KR 348
IWDG_PR 348
IWDG_RLR 349
IWDG_SR 350

P

PWR_CR 54
PWR_CSR 56

R

RCC_AHBENR 84
RCC_APB1ENR 87
RCC_APB1RSTR 82
RCC_APB2ENR 85
RCC_APB2RSTR 80
RCC_BDCR 89
RCC_CFGR 75
RCC_CIR 78
RCC_CR 74
RCC_CSR 90
RTC_ALRH 344
RTC_ALRL 344
RTC_CNTH 343
RTC_CNTL 343
RTC_CRH 339
RTC_CRL 340
RTC_DIVH 342
RTC_DIVL 342
RTC_PRLH 341
RTC_PRLL 342

S

SDIO_CLKCR 444
SDIO_DCOUNT 449
SDIO_DCTRL 448
SDIO_DLEN 447
SDIO_DTIMER 447
SDIO_FIFO 456
SDIO_FIFOCNT 455
SDIO_ICR 451
SDIO_MASK 453
SDIO_POWER 443
SDIO_RESPCMD 446
SDIO_RESPx 446
SDIO_STA 450
SPI_CR1 557
SPI_CR2 560
SPI_CRCPR 563
SPI_DR 562
SPI_I2SCFGR 564
SPI_I2SPR 566
SPI_RXCRCR 563
SPI_SR 561
SPI_TXCRCR 564

T

TIMx_ARR 317, 332
TIMx_BDTR 260
TIMx_CCER 254, 315
TIMx_CCMR1 . . . . . .250, 310
TIMx_CCMR2 . . . . . .253, 314
TIMx_CCR1 . . . . . .258, 317
TIMx_CCR2 . . . . . .259, 317
TIMx_CCR3 . . . . . .259, 318
TIMx_CCR4 . . . . . .260, 318
TIMx_CNT 257, 316, 332
TIMx_CR1 . . . . . .240, 302, 328
TIMx_CR2 . . . . . .241, 303, 330
TIMx_DCR 262, 319
TIMx_DIER 245, 307, 330
TIMx_DMAR 264, 320
TIMx_EGR 249, 309, 331
TIMx_PSC 257, 316, 332
TIMx_RCR 258
TIMx_SMCR 243, 304
TIMx_SR 246, 308, 331

U

USART_BRR 628
USART_CR1 629
USART_CR2 631
USART_CR3 633
USART_DR 628
USART_GTPR 634
USART_SR 625
USB_ADDRn_RX 485
USB_ADDRn_TX 484
USB_BTABLE 479
USB_CNTR 473
USB_COUNTn_RX 485
USB_COUNTn_TX 484
USB_DADDR 479
USB_EPnR 480
USB_FNR 478
USB_ISTR 475

W

WWDG_CFR 355
WWDG_CR 355
WWDG_SR 356

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2008 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com