Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32F030x4/x6/x8/xC and STM32F070x6/xB microcontroller memory and peripherals.

It applies to STM32F030x4/x6/x8/xC and STM32F070x6/xB devices.

For the purpose of this manual, STM32F030x4/x6/x8/xC and STM32F070x6/xB microcontrollers are referred to as STM32F0x0.

The STM32F0x0 is a family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, mechanical and electrical device characteristics, please refer to the corresponding datasheet.

For information on the Arm® Cortex®-M0 core, please refer to the Arm® Cortex®-M0 technical reference manual.

STM32F030x4/x6/x8/xC and STM32F070x6/xB microcontrollers include ST state-of-the-art patented technology.

Related documents

- Arm® Cortex®-M0 technical reference manual, available from Arm website at www.arm.com
- STM32F0xx Cortex-M0 programming manual (PM0215)
- STM32F030x4/x6/x8/xC and STM32F070x6/xB datasheets available from STMicroelectronics website at www.st.com
- STM32F030x4/x6/x8/xC and STM32F070x6/xB errata sheets available from STMicroelectronics website at www.st.com
## Contents

1 **Documentation conventions** ................................................. 33  
  1.1 General information ................................................. 33  
  1.2 List of abbreviations for registers ................................. 33  
  1.3 Glossary ............................................................... 34  
  1.4 Availability of peripherals ........................................... 34  

2 **System and memory overview** ............................................ 35  
  2.1 System architecture .................................................... 35  
  2.2 Memory organization .................................................... 37  
    2.2.1 Introduction ..................................................... 37  
    2.2.2 Memory map and register boundary addresses ............... 38  
  2.3 Embedded SRAM ......................................................... 42  
  2.4 Flash memory overview ............................................... 43  
  2.5 Boot configuration ..................................................... 44  

3 **Embedded flash memory** .................................................. 46  
  3.1 Flash main features .................................................... 46  
  3.2 Flash memory functional description ............................... 46  
    3.2.1 Flash memory organization ................................... 46  
    3.2.2 Flash program and erase operations .......................... 49  
  3.3 Memory protection ..................................................... 56  
    3.3.1 Read protection .................................................. 56  
    3.3.2 Write protection ................................................ 58  
    3.3.3 Option byte write protection ................................ 58  
  3.4 Flash interrupts ....................................................... 58  
  3.5 Flash register description ........................................... 59  
    3.5.1 Flash access control register (FLASH_ACR) ............... 59  
    3.5.2 Flash key register (FLASH_KEYR) ............................. 59  
    3.5.3 Flash option key register (FLASH_OPTKEYR) .............. 60  
    3.5.4 Flash status register (FLASH_SR) ............................ 61  
    3.5.5 Flash control register (FLASH_CR) ........................... 61  
    3.5.6 Flash address register (FLASH_AR) ........................... 63  
    3.5.7 Flash Option byte register (FLASH_OBR) .................... 63
3.5.8 Write protection register (FLASH_WRPR) ......................... 64
3.5.9 Flash register map .............................................. 65

4 Option bytes .......................................................... 66
4.1 Option byte description ............................................. 67
4.1.1 User and read protection option byte ......................... 67
4.1.2 User data option byte ......................................... 68
4.1.3 Write protection option byte ................................ 68
4.1.4 Option byte map .................................................. 69

5 Cyclic redundancy check calculation unit (CRC) ...................... 70
5.1 Introduction .......................................................... 70
5.2 CRC main features .................................................... 70
5.3 CRC functional description ......................................... 71
5.3.1 CRC block diagram .............................................. 71
5.3.2 CRC internal signals .......................................... 71
5.3.3 CRC operation .................................................... 71
5.4 CRC registers ........................................................ 72
5.4.1 CRC data register (CRC_DR) ................................ 72
5.4.2 CRC independent data register (CRC_IDR) .................. 73
5.4.3 CRC control register (CRC_CR) ............................... 73
5.4.4 CRC initial value (CRC_INIT) ................................. 74
5.4.5 CRC register map ................................................ 75

6 Power control (PWR) .................................................. 76
6.1 Power supplies ..................................................... 76
6.1.1 Independent A/D converter supply and reference voltage .. 76
6.1.2 Voltage regulator ................................................. 77
6.2 Power supply supervisor ............................................ 77
6.2.1 Power on reset (POR) / power down reset (PDR) .......... 77
6.3 Low-power modes ................................................... 78
6.3.1 Slowing down system clocks .................................. 79
6.3.2 Peripheral clock gating ....................................... 80
6.3.3 Sleep mode ....................................................... 80
6.3.4 Stop mode ......................................................... 81
6.3.5 Standby mode ..................................................... 83
Contents

6.3.6 RTC wakeup from low-power mode .................................... 84
6.4 Power control registers .................................................. 85
6.4.1 Power control register (PWR_CR) .................................... 85
6.4.2 Power control/status register (PWR_CSR) ......................... 86
6.4.3 PWR register map .................................................... 87

7 Reset and clock control (RCC) ............................................. 88
7.1 Reset ................................................................. 88
7.1.1 Power reset ....................................................... 88
7.1.2 System reset ..................................................... 88
7.1.3 RTC domain reset ................................................ 89
7.2 Clocks ............................................................... 90
7.2.1 HSE clock ....................................................... 93
7.2.2 HSI clock ....................................................... 94
7.2.3 PLL ............................................................. 95
7.2.4 LSE clock ....................................................... 95
7.2.5 LSI clock ....................................................... 96
7.2.6 System clock (SYSCLK) selection ................................ 96
7.2.7 Clock security system (CSS) .................................... 96
7.2.8 ADC clock ....................................................... 97
7.2.9 RTC clock ....................................................... 97
7.2.10 Independent watchdog clock .................................... 97
7.2.11 Clock-out capability ............................................ 97
7.2.12 Internal/external clock measurement with TIM14 ............ 98
7.3 Low-power modes ..................................................... 99
7.4 RCC registers ........................................................ 100
7.4.1 Clock control register (RCC_CR) ................................ 100
7.4.2 Clock configuration register (RCC_CFGR) ...................... 101
7.4.3 Clock interrupt register (RCC_CIR) ............................ 104
7.4.4 APB peripheral reset register 2 (RCC_APB2RSTR) ................ 107
7.4.5 APB peripheral reset register 1 (RCC_APB1RSTR) ............. 108
7.4.6 AHB peripheral clock enable register (RCC_AHBENR) ........ 110
7.4.7 APB peripheral clock enable register 2 (RCC_APB2ENR) .... 111
7.4.8 APB peripheral clock enable register 1 (RCC_APB1ENR) .... 113
7.4.9 RTC domain control register (RCC_BDCR) .................... 115
7.4.10 Control/status register (RCC_CSR) ........................... 116
7.4.11 AHB peripheral reset register (RCC_AHBRESTR) ............. 118
7.4.12 Clock configuration register 2 (RCC_CFGR2) ........................................ 119
7.4.13 Clock configuration register 3 (RCC_CFGR3) ........................................ 120
7.4.14 Clock control register 2 (RCC_CR2) ...................................................... 121
7.4.15 RCC register map .................................................................................. 123

8 General-purpose I/Os (GPIO) ................................................................. 125
8.1 Introduction ............................................................................................... 125
8.2 GPIO main features .................................................................................. 125
8.3 GPIO functional description ...................................................................... 125
  8.3.1 General-purpose I/O (GPIO) ................................................................. 127
  8.3.2 I/O pin alternate function multiplexer and mapping .............................. 127
  8.3.3 I/O port control registers ................................................................. 128
  8.3.4 I/O port data registers ....................................................................... 128
  8.3.5 I/O data bitwise handling .................................................................. 128
  8.3.6 GPIO locking mechanism ................................................................... 129
  8.3.7 I/O alternate function input/output ..................................................... 129
  8.3.8 External interrupt/wake-up lines .......................................................... 129
  8.3.9 Input configuration ............................................................................. 130
  8.3.10 Output configuration ........................................................................ 130
  8.3.11 Alternate function configuration ...................................................... 131
  8.3.12 Analog configuration ........................................................................ 132
8.3.13 Using the HSE or LSE oscillator pins as GPIOs ................................. 133
8.3.14 Using the GPIO pins in the RTC supply domain .................................. 133
8.4 GPIO registers ......................................................................................... 134
  8.4.1 GPIO port mode register (GPIOx_MODER) (x = A to D, F) ................. 134
  8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A to D, F) ....... 134
  8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A to D, F) ... 135
  8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A to D, F) ... 135
  8.4.5 GPIO port input data register (GPIOx_IDR) (x = A to D, F) ............... 136
  8.4.6 GPIO port output data register (GPIOx_ODR) (x = A to D, F) .......... 136
  8.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A to D, F) ......... 137
8.4.8 GPIO port configuration lock register (GPIOx_LCKR)  
(x = A to B) ................................................................. 137
8.4.9 GPIO alternate function low register (GPIOx_AFRL)  
(x = A to D) ................................................................. 138
8.4.10 GPIO alternate function high register (GPIOx_AFRH)  
(x = A to D, F) ............................................................. 139
8.4.11 GPIO port bit reset register (GPIOx_BRR) (x = A to D, F) .......... 139
8.4.12 GPIO register map .................................................. 140

9 System configuration controller (SYSCFG) ......................... 142
  9.1 SYSCFG registers ..................................................... 142
    9.1.1 SYSCFG configuration register 1 (SYSCFG_CFGR1) ............. 142
    9.1.2 SYSCFG external interrupt configuration register 1  
          (SYSCFG_EXTICR1) ........................................ 144
    9.1.3 SYSCFG external interrupt configuration register 2  
          (SYSCFG_EXTICR2) ........................................ 145
    9.1.4 SYSCFG external interrupt configuration register 3  
          (SYSCFG_EXTICR3) ........................................ 145
    9.1.5 SYSCFG external interrupt configuration register 4  
          (SYSCFG_EXTICR4) ........................................ 146
    9.1.6 SYSCFG configuration register 2 (SYSCFG_CFGR2) .......... 147
    9.1.7 SYSCFG register maps ......................................... 148

10 Direct memory access controller (DMA) .......................... 149
  10.1 Introduction ...................................................... 149
  10.2 DMA main features ............................................... 149
  10.3 DMA implementation ............................................. 150
    10.3.1 DMA .......................................................... 150
    10.3.2 DMA request mapping ...................................... 150
  10.4 DMA functional description .................................... 153
    10.4.1 DMA block diagram ......................................... 153
    10.4.2 DMA transfers ............................................... 154
    10.4.3 DMA arbitration ............................................. 155
    10.4.4 DMA channels ............................................... 155
    10.4.5 DMA data width, alignment and endianness ............... 159
    10.4.6 DMA error management ..................................... 160
  10.5 DMA interrupts .................................................. 161
  10.6 DMA registers ................................................... 161
10.6.1 DMA interrupt status register (DMA_ISR) ............................................. 161
10.6.2 DMA interrupt flag clear register (DMA_IFCR) ................................. 163
10.6.3 DMA channel x configuration register (DMA_CCRx) ......................... 164
10.6.4 DMA channel x number of data to transfer register (DMA_CNDTRx) . 166
10.6.5 DMA channel x peripheral address register (DMA_CPARx) ............. 167
10.6.6 DMA channel x memory address register (DMA_CMARx) ................. 168
10.6.7 DMA channel selection register (DMA_CSELR) ............................. 168
10.6.8 DMA register map ........................................................................ 169

11 Interrupts and events ............................................................................. 171
11.1 Nested vectored interrupt controller (NVIC) ......................................... 171
  11.1.1 NVIC main features ....................................................................... 171
  11.1.2 SysTick calibration value register .................................................. 171
  11.1.3 Interrupt and exception vectors ..................................................... 171
11.2 Extended interrupts and events controller (EXTI) ............................... 173
  11.2.1 Main features ............................................................................... 173
  11.2.2 Block diagram ........................................................................... 174
  11.2.3 Event management ...................................................................... 174
  11.2.4 Functional description .................................................................. 174
  11.2.5 External and internal interrupt/event line mapping ......................... 176
11.3 EXTI registers ...................................................................................... 177
  11.3.1 Interrupt mask register (EXTI_IMR) ............................................ 177
  11.3.2 Event mask register (EXTI_EMR) ................................................ 177
  11.3.3 Rising trigger selection register (EXTI_RTSR) ........................... 177
  11.3.4 Falling trigger selection register (EXTI_FTSR) ............................ 178
  11.3.5 Software interrupt event register (EXTI_SWIER) ....................... 179
  11.3.6 Pending register (EXTI_PR) ........................................................ 179
  11.3.7 EXTI register map ...................................................................... 181

12 Analog-to-digital converter (ADC) ......................................................... 182
12.1 Introduction ......................................................................................... 182
12.2 ADC main features ............................................................................ 183
12.3 ADC functional description .............................................................. 184
  12.3.1 ADC pins and internal signals ....................................................... 184
  12.3.2 Calibration (ADCAL) .................................................................. 185
  12.3.3 ADC on-off control (ADEN, ADDIS, ADRDY) ......................... 186
  12.3.4 ADC clock (CKMODE) ............................................................... 188
12.3.5 Configuring the ADC ........................................ 189
12.3.6 Channel selection (CHSEL, SCANDIR) ..................... 189
12.3.7 Programmable sampling time (SMP) ......................... 190
12.3.8 Single conversion mode (CONT = 0) ....................... 190
12.3.9 Continuous conversion mode (CONT = 1) ................... 191
12.3.10 Starting conversions (ADSTART) ......................... 191
12.3.11 Timings .................................................. 192
12.3.12 Stopping an ongoing conversion (ADSTP) ............... 193

12.4 Conversion on external trigger and trigger polarity (EXTSEL, EX TEN) ........................................ 193
12.4.1 Discontinuous mode (DISCEN) ............................. 194
12.4.2 Programmable resolution (RES) - Fast conversion mode ........................ 194
12.4.3 End of conversion, end of sampling phase (EOC, EOSMP flags) ........................ 195
12.4.4 End of conversion sequence (EOS flag) .................... 195
12.4.5 Example timing diagrams (single/continuous modes hardware/software triggers) ........................ 196

12.5 Data management ............................................. 198
12.5.1 Data register and data alignment (ADC_DR, ALIGN) .... 198
12.5.2 ADC overrun (OVR, OVRMOD) .............................. 198
12.5.3 Managing a sequence of data converted without using the DMA ........ 199
12.5.4 Managing converted data without using the DMA without overrun ........ 199
12.5.5 Managing converted data using the DMA ............... 199

12.6 Low-power features ........................................... 201
12.6.1 Wait mode conversion ........................................ 201
12.6.2 Auto-off mode (AUTOFF) .................................. 202

12.7 Analog window watchdog ....................................... 203
12.7.1 Description of the analog watchdog ......................... 203
12.7.2 ADC_AWD1_OUT output signal generation ................ 204
12.7.3 Analog watchdog threshold control ......................... 206

12.8 Temperature sensor and internal reference voltage .......... 207

12.9 ADC interrupts .............................................. 210

12.10 ADC registers ................................................ 211
12.10.1 ADC interrupt and status register (ADC_ISR) .......... 211
12.10.2 ADC interrupt enable register (ADC_IER) ............... 212
12.10.3 ADC control register (ADC_CR) ......................... 214
12.10.4 ADC configuration register 1 (ADC_CFGR1) .......... 216
12.10.5 ADC configuration register 2 (ADC_CFGR2) .......... 220
12.10.6 ADC sampling time register (ADC_SMPR) ............................................. 220
12.10.7 ADC watchdog threshold register (ADC_TR) ....................................... 221
12.10.8 ADC channel selection register (ADC_CHSELR) ..................................... 221
12.10.9 ADC data register (ADC_DR) ................................................................. 222
12.10.10 ADC common configuration register (ADC_CCR) .................................. 223
12.11 ADC register map ...................................................................................... 223

13 Advanced-control timers (TIM1) ................................................................. 225
13.1 TIM1 introduction ...................................................................................... 225
13.2 TIM1 main features ..................................................................................... 225
13.3 TIM1 functional description ....................................................................... 227
13.3.1 Time-base unit ......................................................................................... 227
13.3.2 Counter modes ......................................................................................... 229
13.3.3 Repetition counter .................................................................................... 239
13.3.4 Clock sources .......................................................................................... 241
13.3.5 Capture/compare channels ................................................................. 244
13.3.6 Input capture mode .................................................................................... 247
13.3.7 PWM input mode ....................................................................................... 248
13.3.8 Forced output mode ................................................................................... 249
13.3.9 Output compare mode .............................................................................. 249
13.3.10 PWM mode ............................................................................................. 250
13.3.11 Complementary outputs and dead-time insertion ................................. 254
13.3.12 Using the break function ........................................................................ 256
13.3.13 Clearing the OCxREF signal on an external event ................................. 259
13.3.14 6-step PWM generation ................................................................. 261
13.3.15 One-pulse mode ....................................................................................... 262
13.3.16 Encoder interface mode ......................................................................... 263
13.3.17 Timer input XOR function ................................................................. 266
13.3.18 Interfacing with Hall sensors ............................................................ 266
13.3.19 TIMx and external trigger synchronization ........................................... 268
13.3.20 Timer synchronization ........................................................................... 271
13.3.21 Debug mode ............................................................................................ 271
13.4 TIM1 registers ............................................................................................. 272
13.4.1 TIM1 control register 1 (TIM1_CR1) ....................................................... 272
13.4.2 TIM1 control register 2 (TIM1_CR2) ....................................................... 273
13.4.3 TIM1 slave mode control register (TIM1_SMCR) ..................................... 275
13.4.4 TIM1 DMA/interrupt enable register (TIM1_DIER) ............................... 278
13.4.5 TIM1 status register (TIM1_SR) ................................. 280
13.4.6 TIM1 event generation register (TIM1_EGR) ..................... 281
13.4.7 TIM1 capture/compare mode register 1 (TIM1_CCMR1) ........ 283
13.4.8 TIM1 capture/compare mode register 2 (TIM1_CCMR2) ........ 286
13.4.9 TIM1 capture/compare enable register (TIM1_CCER) .......... 288
13.4.10 TIM1 counter (TIM1_CNT) ........................................... 291
13.4.11 TIM1 prescaler (TIM1_PSC) ........................................ 292
13.4.12 TIM1 auto-reload register (TIM1_ARR) .......................... 292
13.4.13 TIM1 repetition counter register (TIM1_RCR) ................. 292
13.4.14 TIM1 capture/compare register 1 (TIM1_CCR1) ............... 293
13.4.15 TIM1 capture/compare register 2 (TIM1_CCR2) ............... 293
13.4.16 TIM1 capture/compare register 3 (TIM1_CCR3) ............... 294
13.4.17 TIM1 capture/compare register 4 (TIM1_CCR4) ............... 295
13.4.18 TIM1 break and dead-time register (TIM1_BDTR) ............. 295
13.4.19 TIM1 DMA control register (TIM1_DCR) ........................ 297
13.4.20 TIM1 DMA address for full transfer (TIM1_DMAR) ............ 298
13.4.21 TIM1 register map .................................................. 299

14 General-purpose timers (TIM3) .......................................... 301
14.1 TIM3 introduction ...................................................... 301
14.2 TIM3 main features .................................................... 301
14.3 TIM3 functional description ...................................... 302
   14.3.1 Time-base unit .................................................. 302
   14.3.2 Counter modes .................................................. 304
   14.3.3 Clock sources .................................................. 315
   14.3.4 Capture/compare channels .................................. 318
   14.3.5 Input capture mode ............................................ 320
   14.3.6 PWM input mode .............................................. 322
   14.3.7 Forced output mode .......................................... 323
   14.3.8 Output compare mode ........................................ 323
   14.3.9 PWM mode ...................................................... 324
   14.3.10 One-pulse mode .............................................. 328
   14.3.11 Clearing the OCxREF signal on an external event .......... 329
   14.3.12 Encoder interface mode .................................... 330
   14.3.13 Timer input XOR function ................................... 332
   14.3.14 Timers and external trigger synchronization .............. 333
   14.3.15 Timer synchronization ...................................... 336
14.3.16 Debug mode ...................................................... 342

14.4 TIM3 registers ...................................................... 343

14.4.1 TIM3 control register 1 (TIM3_CR1) ......................... 343
14.4.2 TIM3 control register 2 (TIM3_CR2) ......................... 345
14.4.3 TIM3 slave mode control register (TIM3_SMCR) .......... 346
14.4.4 TIM3 DMA/Interrupt enable register (TIM3_DIER) .... 348
14.4.5 TIM3 status register (TIM3_SR) ......................... 349
14.4.6 TIM3 event generation register (TIM3_EGR) ............ 352
14.4.7 TIM3 capture/compare mode register 1 (TIM3_CCMR1) 353
14.4.8 TIM3 capture/compare mode register 2 (TIM3_CCMR2) 356
14.4.9 TIM3 capture/compare enable register (TIM3_CCER) .... 357
14.4.10 TIM3 counter (TIM3_CNT) ............................... 359
14.4.11 TIM3 prescaler (TIM3_PSC) .............................. 359
14.4.12 TIM3 auto-reload register (TIM3_ARR) .................. 360
14.4.13 TIM3 capture/compare register 1 (TIM3_CCR1) ....... 360
14.4.14 TIM3 capture/compare register 2 (TIM3_CCR2) ....... 360
14.4.15 TIM3 capture/compare register 3 (TIM3_CCR3) ....... 360
14.4.16 TIM3 capture/compare register 4 (TIM3_CCR4) ....... 362
14.4.17 TIM3 DMA control register (TIM3_DCR) ............. 362
14.4.18 TIM3 DMA address for full transfer (TIM3_DMAR) ..... 363
14.4.19 TIM3 register map ........................................ 365

15 Basic timer (TIM6/TIM7) ......................................... 367

15.1 TIM6/TIM7 introduction ...................................... 367
15.2 TIM6/TIM7 main features .................................... 367
15.3 TIM6/TIM7 functional description ........................... 368
15.3.1 Time-base unit ............................................. 368
15.3.2 Counter modes ............................................. 370
15.3.3 Clock source .............................................. 374
15.3.4 Debug mode .............................................. 374
15.4 TIM6/TIM7 registers .......................................... 375
15.4.1 TIM6/TIM7 control register 1 (TIMx_CR1) ............ 375
15.4.2 TIM6/TIM7 DMA/Interrupt enable register (TIMx_DIER) 376
15.4.3 TIM6/TIM7 status register (TIMx_SR) ................. 377
15.4.4 TIM6/TIM7 event generation register (TIMx_EGR) .... 377
15.4.5 TIM6/TIM7 counter (TIMx_CNT) ......................... 377
15.4.6 TIM6/TIM7 prescaler (TIMx_PSC) ...................... 378
15.4.7 TIM6/TIM7 auto-reload register (TIMx.ARR) .............................. 378
15.4.8 TIM6/TIM7 register map ......................................................... 379

16 General-purpose timer (TIM14) .............................................. 380
16.1 TIM14 introduction ............................................................... 380
16.2 TIM14 main features ............................................................ 380
16.3 TIM14 functional description .................................................. 381
16.3.1 Time-base unit ................................................................. 381
16.3.2 Counter operation ............................................................. 382
16.3.3 Clock source ................................................................. 385
16.3.4 Capture/compare channels ................................................ 385
16.3.5 Input capture mode ........................................................... 387
16.3.6 Forced output mode .......................................................... 388
16.3.7 Output compare mode ....................................................... 388
16.3.8 PWM mode ................................................................. 390
16.3.9 Debug mode ................................................................. 391
16.4 TIM14 registers ................................................................. 391
16.4.1 TIM14 control register 1 (TIM14.CR1) .................................. 391
16.4.2 TIM14 interrupt enable register (TIM14.DIER) ....................... 392
16.4.3 TIM14 status register (TIM14.SR) ........................................ 393
16.4.4 TIM14 event generation register (TIM14.EGR) ....................... 393
16.4.5 TIM14 capture/compare mode register 1 [alternate] (TIM14.CCMR1) .... 394
16.4.6 TIM14 capture/compare mode register 1 [alternate] (TIM14.CCMR1) .... 395
16.4.7 TIM14 capture/compare enable register (TIM14.CCER) ............. 396
16.4.8 TIM14 counter (TIM14.CNT) .............................................. 397
16.4.9 TIM14 prescaler (TIM14.PSC) ............................................ 398
16.4.10 TIM14 auto-reload register (TIM14.ARR) ............................ 398
16.4.11 TIM14 capture/compare register 1 (TIM14.CCR1) .................. 398
16.4.12 TIM14 option register (TIM14.OR) ..................................... 399
16.4.13 TIM14 register map ....................................................... 399

17 General-purpose timers (TIM15/16/17) ................................. 401
17.1 TIM15/16/17 introduction ..................................................... 401
17.2 TIM15 main features ............................................................ 401
17.3 TIM16 and TIM17 main features ........................................... 403
17.4 TIM15/16/17 functional description ....................................... 404
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.4.1</td>
<td>Time-base unit</td>
<td>404</td>
</tr>
<tr>
<td>17.4.2</td>
<td>Counter operation</td>
<td>406</td>
</tr>
<tr>
<td>17.4.3</td>
<td>Repetition counter</td>
<td>410</td>
</tr>
<tr>
<td>17.4.4</td>
<td>Clock sources</td>
<td>411</td>
</tr>
<tr>
<td>17.4.5</td>
<td>Capture/compare channels</td>
<td>413</td>
</tr>
<tr>
<td>17.4.6</td>
<td>Input capture mode</td>
<td>416</td>
</tr>
<tr>
<td>17.4.7</td>
<td>PWM input mode (only for TIM15)</td>
<td>417</td>
</tr>
<tr>
<td>17.4.8</td>
<td>Forced output mode</td>
<td>417</td>
</tr>
<tr>
<td>17.4.9</td>
<td>Output compare mode</td>
<td>418</td>
</tr>
<tr>
<td>17.4.10</td>
<td>PWM mode</td>
<td>419</td>
</tr>
<tr>
<td>17.4.11</td>
<td>Complementary outputs and dead-time insertion</td>
<td>420</td>
</tr>
<tr>
<td>17.4.12</td>
<td>Using the break function</td>
<td>422</td>
</tr>
<tr>
<td>17.4.13</td>
<td>One-pulse mode</td>
<td>425</td>
</tr>
<tr>
<td>17.4.14</td>
<td>TIM15 external trigger synchronization</td>
<td>426</td>
</tr>
<tr>
<td>17.4.15</td>
<td>Timer synchronization (TIM15)</td>
<td>429</td>
</tr>
<tr>
<td>17.4.16</td>
<td>Debug mode</td>
<td>429</td>
</tr>
<tr>
<td>17.5</td>
<td>TIM15 registers</td>
<td>430</td>
</tr>
<tr>
<td>17.5.1</td>
<td>TIM15 control register 1 (TIM15_CR1)</td>
<td>430</td>
</tr>
<tr>
<td>17.5.2</td>
<td>TIM15 control register 2 (TIM15_CR2)</td>
<td>431</td>
</tr>
<tr>
<td>17.5.3</td>
<td>TIM15 slave mode control register (TIM15_SMCR)</td>
<td>432</td>
</tr>
<tr>
<td>17.5.4</td>
<td>TIM15 DMA/interrupt enable register (TIM15_DIER)</td>
<td>434</td>
</tr>
<tr>
<td>17.5.5</td>
<td>TIM15 status register (TIM15_SR)</td>
<td>435</td>
</tr>
<tr>
<td>17.5.6</td>
<td>TIM15 event generation register (TIM15_EGR)</td>
<td>437</td>
</tr>
<tr>
<td>17.5.7</td>
<td>TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)</td>
<td>438</td>
</tr>
<tr>
<td>17.5.8</td>
<td>TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)</td>
<td>439</td>
</tr>
<tr>
<td>17.5.9</td>
<td>TIM15 capture/compare enable register (TIM15_CCER)</td>
<td>441</td>
</tr>
<tr>
<td>17.5.10</td>
<td>TIM15 counter (TIM15_CNT)</td>
<td>444</td>
</tr>
<tr>
<td>17.5.11</td>
<td>TIM15 prescaler (TIM15_PSC)</td>
<td>444</td>
</tr>
<tr>
<td>17.5.12</td>
<td>TIM15 auto-reload register (TIM15_ARR)</td>
<td>444</td>
</tr>
<tr>
<td>17.5.13</td>
<td>TIM15 repetition counter register (TIM15_RCR)</td>
<td>445</td>
</tr>
<tr>
<td>17.5.14</td>
<td>TIM15 capture/compare register 1 (TIM15_CCR1)</td>
<td>445</td>
</tr>
<tr>
<td>17.5.15</td>
<td>TIM15 capture/compare register 2 (TIM15_CCR2)</td>
<td>445</td>
</tr>
<tr>
<td>17.5.16</td>
<td>TIM15 break and dead-time register (TIM15_BDTR)</td>
<td>446</td>
</tr>
<tr>
<td>17.5.17</td>
<td>TIM15 DMA control register (TIM15_DCR)</td>
<td>448</td>
</tr>
<tr>
<td>17.5.18</td>
<td>TIM15 DMA address for full transfer (TIM15_DMAR)</td>
<td>449</td>
</tr>
<tr>
<td>17.5.19</td>
<td>TIM15 register map</td>
<td>449</td>
</tr>
<tr>
<td>17.6</td>
<td>TIM16/TIM17 registers</td>
<td>450</td>
</tr>
</tbody>
</table>
17.6.1 TIMx control register 1 (TIMx_CR1)(x = 16 to 17) .......................... 450
17.6.2 TIMx control register 2 (TIMx_CR2)(x = 16 to 17) ..................... 452
17.6.3 TIMx DMA/interrupt enable register (TIMx_DIER)(x = 16 to 17) ..... 452
17.6.4 TIMx status register (TIMx_SR)(x = 16 to 17) .......................... 453
17.6.5 TIMx event generation register (TIMx_EGR)(x = 16 to 17) .......... 454
17.6.6 TIMx capture/compare mode register 1 [alternate] 
(TIMx_CCMR1)(x = 16 to 17) ................................................. 455
17.6.7 TIMx capture/compare mode register 1 [alternate] 
(TIMx_CCMR1)(x = 16 to 17) ................................................. 456
17.6.8 TIMx capture/compare enable register (TIMx_CCR)(x = 16 to 17) .. 458
17.6.9 TIMx counter (TIMx_CNT)(x = 16 to 17) ................................ 461
17.6.10 TIMx prescaler (TIMx_PSC)(x = 16 to 17) ........................... 461
17.6.11 TIMx auto-reload register (TIMx_ARR)(x = 16 to 17) ............... 461
17.6.12 TIMx repetition counter register (TIMx_RCR)(x = 16 to 17) ........ 462
17.6.13 TIMx capture/compare register 1 (TIMx_CCR1)(x = 16 to 17) ...... 462
17.6.14 TIMx break and dead-time register (TIMx_BDTR)(x = 16 to 17) ... 462
17.6.15 TIMx DMA control register (TIMx_DCR)(x = 16 to 17) ............. 464
17.6.16 TIMx DMA address for full transfer (TIMx_DMAR)(x = 16 to 17) ... 465
17.6.17 TIM16/TIM17 register map ............................................. 466

18  Infrared interface (IRTIM) .................................................. 468

19  Independent watchdog (IWDG) ............................................. 469

19.1 Introduction .............................................................. 469
19.2 IWDG main features .................................................... 469
19.3 IWDG functional description ............................................ 469
  19.3.1 IWDG block diagram ............................................. 469
  19.3.2 Window option .................................................... 470
  19.3.3 Hardware watchdog .............................................. 471
  19.3.4 Register access protection .................................... 471
  19.3.5 Debug mode ....................................................... 471
19.4 IWDG registers ......................................................... 472
  19.4.1 IWDG key register (IWDG_KR) ................................ 472
  19.4.2 IWDG prescaler register (IWDG_PR) ......................... 473
  19.4.3 IWDG reload register (IWDG_RLR) ............................ 474
  19.4.4 IWDG status register (IWDG_SR) ............................. 475
  19.4.5 IWDG window register (IWDG_WINR) ....................... 476
20 System window watchdog (WWDG) ................................................. 478
  20.1 Introduction ................................................................. 478
  20.2 WWDG main features ..................................................... 478
  20.3 WWDG functional description .......................................... 478
    20.3.1 WWDG block diagram .............................................. 479
    20.3.2 Enabling the watchdog ............................................ 479
    20.3.3 Controlling the down-counter .................................... 479
    20.3.4 How to program the watchdog timeout ......................... 479
    20.3.5 Debug mode .......................................................... 480
  20.4 WWDG interrupts .......................................................... 481
  20.5 WWDG registers ............................................................ 481
    20.5.1 WWDG control register (WWDG_CR) ............................. 481
    20.5.2 WWDG configuration register (WWDG_CFR) .................... 482
    20.5.3 WWDG status register (WWDG_SR) ............................... 482
    20.5.4 WWDG register map ................................................ 483

21 Real-time clock (RTC) .......................................................... 484
  21.1 Introduction ................................................................. 484
  21.2 RTC main features ........................................................ 485
  21.3 RTC implementation ...................................................... 485
  21.4 RTC functional description ............................................ 486
    21.4.1 RTC block diagram ............................................... 486
    21.4.2 GPIOs controlled by the RTC .................................... 488
    21.4.3 Clock and prescalers .............................................. 489
    21.4.4 Real-time clock and calendar ................................... 490
    21.4.5 Programmable alarm ............................................... 490
    21.4.6 Periodic auto-wake-up ............................................. 491
    21.4.7 RTC initialization and configuration ......................... 491
    21.4.8 Reading the calendar .............................................. 493
    21.4.9 Resetting the RTC ................................................. 494
    21.4.10 RTC synchronization ............................................. 494
    21.4.11 RTC reference clock detection ................................. 495
    21.4.12 RTC smooth digital calibration ............................... 496
    21.4.13 Time-stamp function ............................................. 498
## Contents

21.4.14 Tamper detection ............................................. 498  
21.4.15 Calibration clock output .................................... 500  
21.4.16 Alarm output .................................................... 500  
21.5 RTC low-power modes ............................................. 501  
21.6 RTC interrupts ..................................................... 501  
21.7 RTC registers ....................................................... 501  
  21.7.1 RTC time register (RTC_TR) .................................. 501  
  21.7.2 RTC date register (RTC_DR) .................................. 502  
  21.7.3 RTC control register (RTC_CR) .............................. 504  
  21.7.4 RTC initialization and status register (RTC_ISR) ........ 507  
  21.7.5 RTC prescaler register (RTC_PRER) ....................... 509  
  21.7.6 RTC wake-up timer register (RTC_WUTR) .................. 510  
  21.7.7 RTC alarm A register (RTC_ALRMAR) ..................... 511  
  21.7.8 RTC write protection register (RTC_WPR) ............... 512  
  21.7.9 RTC sub second register (RTC_SSR) ....................... 512  
  21.7.10 RTC shift control register (RTC_SHIFTR) ............. 513  
  21.7.11 RTC timestamp time register (RTC_TSTR) ............. 514  
  21.7.12 RTC timestamp date register (RTC_TSDR) ............. 515  
  21.7.13 RTC time-stamp sub second register (RTC_TSSSR) .... 516  
  21.7.14 RTC calibration register (RTC_CALR) ................... 517  
  21.7.15 RTC tamper and alternate function configuration register (RTC_TAFCR) .............................. 518  
  21.7.16 RTC alarm A sub second register (RTC_ALRMASSR) .... 521  
  21.7.17 RTC register map ............................................. 521  

22 Inter-integrated circuit (I2C) interface .......................... 524  
  22.1 Introduction ..................................................... 524  
  22.2 I2C main features ............................................... 524  
  22.3 I2C implementation ............................................. 525  
  22.4 I2C functional description ..................................... 525  
    22.4.1 I2C block diagram .......................................... 526  
    22.4.2 I2C2 block diagram ........................................ 526  
    22.4.3 I2C pins and internal signals ............................. 527  
    22.4.4 I2C clock requirements .................................... 527  
    22.4.5 Mode selection .............................................. 528  
    22.4.6 I2C initialization .......................................... 529  
    22.4.7 Software reset .............................................. 533  

16/775 RM0360 Rev 5
22.4.8 Data transfer .................................................. 534
22.4.9 I2C slave mode ............................................. 536
22.4.10 I2C master mode .......................................... 545
22.4.11 I2C_TIMINGR register configuration examples ............. 556
22.4.12 SMBus specific features .................................. 558
22.4.13 SMBus initialization ....................................... 561
22.4.14 SMBus: I2C_TIMEOUTR register configuration examples ........ 563
22.4.15 SMBus slave mode ........................................ 563
22.4.16 Error conditions ......................................... 570
22.4.17 DMA requests ............................................ 572
22.4.18 Debug mode ............................................. 573

22.5 I2C low-power modes ....................................... 573
22.6 I2C interrupts ............................................... 574
22.7 I2C registers ................................................ 575
  22.7.1 I2C control register 1 (I2C_CR1) ............................. 575
  22.7.2 I2C control register 2 (I2C_CR2) ............................. 577
  22.7.3 I2C own address 1 register (I2C_OAR1) ...................... 579
  22.7.4 I2C own address 2 register (I2C_OAR2) ...................... 580
  22.7.5 I2C timing register (I2C_TIMINGR) ......................... 581
  22.7.6 I2C timeout register (I2C_TIMEOUTR) ...................... 582
  22.7.7 I2C interrupt and status register (I2C_ISR) ................. 583
  22.7.8 I2C interrupt clear register (I2C_ICR) ...................... 585
  22.7.9 I2C PEC register (I2C_PECR) .............................. 586
  22.7.10 I2C receive data register (I2C_RXDR) ....................... 587
  22.7.11 I2C transmit data register (I2C_TXDR) ....................... 587
  22.7.12 I2C register map ........................................ 588

23 Universal synchronous/asynchronous receiver transmitter (USART/UART) ............................... 590
  23.1 Introduction .................................................. 590
  23.2 USART main features ....................................... 590
  23.3 USART implementation ..................................... 592
  23.4 USART functional description .............................. 593
    23.4.1 USART character description ............................. 594
    23.4.2 USART transmitter ..................................... 595
    23.4.3 USART receiver ........................................ 598
23.4.4 USART baud rate generation ........................................... 604
23.4.5 Tolerance of the USART receiver to clock deviation .............. 606
23.4.6 USART auto baud rate detection ...................................... 607
23.4.7 Multiprocessor communication using USART ......................... 607
23.4.8 USART parity control .................................................... 610
23.4.9 USART synchronous mode .............................................. 610
23.4.10 USART Single-wire Half-duplex communication ...................... 613
23.4.11 USART continuous communication in DMA mode ................. 613
23.4.12 RS232 hardware flow control and RS485 driver enable using USART ....................................................... 616
23.5 USART in low-power modes .............................................. 618
23.6 USART interrupts .......................................................... 619
23.7 USART registers ............................................................ 620
   23.7.1 USART control register 1 (USART_CR1) .......................... 620
   23.7.2 USART control register 2 (USART_CR2) .......................... 623
   23.7.3 USART control register 3 (USART_CR3) .......................... 626
   23.7.4 USART baud rate register (USART_BRR) ......................... 628
   23.7.5 USART receiver timeout register (USART_RTOR) ............... 628
   23.7.6 USART request register (USART_RQR) ............................. 629
   23.7.7 USART interrupt and status register (USART_ISR) .............. 630
   23.7.8 USART interrupt flag clear register (USART_ICR) ............... 633
   23.7.9 USART receive data register (USART_RDR) ....................... 634
   23.7.10 USART transmit data register (USART_TDR) ..................... 635
   23.7.11 USART register map .................................................. 635

24 Serial peripheral interface (SPI) ......................................... 637
   24.1 Introduction .............................................................. 637
   24.2 SPI main features ........................................................ 637
   24.3 SPI implementation ...................................................... 638
   24.4 SPI functional description .............................................. 638
      24.4.1 General description .................................................. 638
      24.4.2 Communications between one master and one slave ........... 639
      24.4.3 Standard multislave communication ............................... 642
      24.4.4 Multimaster communication ........................................ 642
      24.4.5 Slave select (NSS) pin management ............................... 643
      24.4.6 Communication formats ............................................. 644
      24.4.7 Configuration of SPI ............................................... 646
24.4.8 Procedure for enabling SPI ........................................... 647
24.4.9 Data transmission and reception procedures ....................... 647
24.4.10 SPI status flags ...................................................... 657
24.4.11 SPI error flags ...................................................... 658
24.4.12 NSS pulse mode ..................................................... 659
24.4.13 TI mode ............................................................. 659
24.4.14 CRC calculation ...................................................... 660
24.5 SPI interrupts .................................................................. 662
24.6 SPI registers .................................................................. 663
  24.6.1 SPI control register 1 (SPIx_CR1) .................................... 663
  24.6.2 SPI control register 2 (SPIx_CR2) .................................... 665
  24.6.3 SPI status register (SPIx_SR) ......................................... 667
  24.6.4 SPI data register (SPIx_DR) ........................................... 668
  24.6.5 SPI CRC polynomial register (SPIx_CRCPR) .................... 669
  24.6.6 SPI Tx CRC register (SPIx_TXCRCR) ............................... 669
  24.6.7 SPI Tx CRC register (SPIx_TXCRCR) ............................... 669
  24.6.8 SPI register map ....................................................... 671

25 Universal serial bus full-speed device interface (USB) ............... 672
  25.1 Introduction .............................................................. 672
  25.2 USB main features ...................................................... 672
  25.3 USB implementation ................................................... 672
  25.4 USB functional description ............................................ 673
    25.4.1 Description of USB blocks .................................... 674
  25.5 Programming considerations ......................................... 675
    25.5.1 Generic USB device programming .............................. 675
    25.5.2 System and power-on reset .................................... 676
    25.5.3 Double-buffered endpoints .................................... 681
    25.5.4 Isochronous transfers ........................................... 683
    25.5.5 Suspend/Resume events ......................................... 684
  25.6 USB and USB SRAM registers ........................................ 687
    25.6.1 Common registers ................................................ 687
    25.6.2 Buffer descriptor table ........................................ 700
    25.6.3 USB register map ................................................ 703

26 Debug support (DBG) ..................................................... 705
26.1 Overview .................................................. 705
26.2 Reference Arm documentation .............................. 706
26.3 Pinout and debug port pins ............................... 706
   26.3.1 SWD port pins ........................................ 707
   26.3.2 SW-DP pin assignment .............................. 707
   26.3.3 Internal pull-up and pull-down on SWD pins ....... 707
26.4 ID codes and locking mechanism .......................... 707
   26.4.1 MCU device ID code .................................. 708
26.5 SWD port ................................................. 708
   26.5.1 SWD protocol introduction .......................... 708
   26.5.2 SWD protocol sequence ............................. 709
   26.5.3 SW-DP state machine (reset, idle states, ID code) 710
   26.5.4 DP and AP read/write accesses ....................... 710
   26.5.5 SW-DP registers ..................................... 711
   26.5.6 SW-AP registers ..................................... 712
26.6 Core debug .............................................. 712
26.7 BPU (Break Point Unit) .................................. 713
   26.7.1 BPU functionality ................................... 713
26.8 DWT (Data Watchpoint) .................................. 713
   26.8.1 DWT functionality ................................... 713
   26.8.2 DWT Program Counter Sample Register ............. 713
26.9 MCU debug component (DBGMCU) ......................... 713
   26.9.1 Debug support for low-power modes .................. 714
   26.9.2 Debug support for timers, watchdog and I²C ........ 714
   26.9.3 Debug MCU configuration register (DBGMCU_CR) .... 715
   26.9.4 Debug MCU APB1 freeze register (DBGMCU_APB1_FZ) 716
   26.9.5 Debug MCU APB2 freeze register (DBGMCU_APB2_FZ) 718
   26.9.6 DBG register map .................................... 718

27 Device electronic signature ................................. 720
   27.1 Flash memory size data register ......................... 720

Appendix A Code examples .................................... 721
   A.1 Introduction .............................................. 721
   A.2 FLASH operation code examples ........................ 721
      A.2.1 Flash memory unlocking sequence ................... 721
A.2.2 Main flash memory programming sequence .......................................................... 721
A.2.3 Page erase sequence .............................................................................................. 722
A.2.4 Mass erase sequence.............................................................................................. 723
A.2.5 Option byte unlocking sequence .......................................................................... 723
A.2.6 Option byte programming sequence ...................................................................... 724
A.2.7 Option byte erasing sequence ................................................................................. 724

A.3 Clock controller code examples ................................................................................. 725
A.3.1 HSE start sequence .............................................................................................. 725
A.3.2 PLL configuration modification ............................................................................. 726
A.3.3 MCO selection ...................................................................................................... 726
A.3.4 Clock measurement configuration with TIM14 ....................................................... 727

A.4 GPIO code examples .................................................................................................. 728
A.4.1 Lock sequence ........................................................................................................ 728
A.4.2 Alternate function selection sequence .................................................................... 728
A.4.3 Analog GPIO configuration ..................................................................................... 729

A.5 DMA code examples .................................................................................................. 729
A.5.1 DMA channel configuration sequence ................................................................... 729

A.6 Interrupts and event code examples .......................................................................... 730
A.6.1 NVIC initialization .................................................................................................. 730
A.6.2 External interrupt selection .................................................................................... 730

A.7 ADC code examples ................................................................................................... 731
A.7.1 ADC calibration ...................................................................................................... 731
A.7.2 ADC enable sequence ............................................................................................ 731
A.7.3 ADC disable sequence ........................................................................................... 732
A.7.4 ADC clock selection ............................................................................................... 732
A.7.5 Single conversion sequence - software trigger ....................................................... 733
A.7.6 Continuous conversion sequence - software trigger ............................................. 733
A.7.7 Single conversion sequence - hardware trigger ..................................................... 734
A.7.8 Continuous conversion sequence - hardware trigger ........................................... 734
A.7.9 DMA one shot mode sequence .............................................................................. 735
A.7.10 DMA circular mode sequence .............................................................................. 735
A.7.11 Wait mode sequence ............................................................................................. 735
A.7.12 Auto Off and no wait mode sequence .................................................................... 736
A.7.13 Auto Off and wait mode sequence ........................................................................ 736
A.7.14 Analog watchdog ................................................................................................. 736
A.7.15 Temperature configuration .................................................................................... 737
A.7.16 Temperature computation .................................................. 737
A.8 Timers ............................................................................ 738
   A.8.1 Upcounter on TI2 rising edge ........................................ 738
   A.8.2 Up counter on each 2 ETR rising edges ............................ 739
   A.8.3 Input capture configuration ........................................... 739
   A.8.4 Input capture data management ....................................... 740
   A.8.5 PWM input configuration .............................................. 741
   A.8.6 PWM input with DMA configuration ............................... 741
   A.8.7 Output compare configuration ......................................... 742
   A.8.8 Edge-aligned PWM configuration example ....................... 742
   A.8.9 Center-aligned PWM configuration example ..................... 743
   A.8.10 ETR configuration to clear OCxREF ................................. 744
   A.8.11 Encoder interface ........................................................ 744
   A.8.12 Reset mode ............................................................... 745
   A.8.13 Gated mode ............................................................... 745
   A.8.14 Trigger mode ............................................................. 746
   A.8.15 External clock mode 2 + trigger mode ............................ 746
   A.8.16 One-Pulse mode ........................................................ 747
   A.8.17 Timer prescaling another timer ...................................... 747
   A.8.18 Timer enabling another timer ....................................... 748
   A.8.19 Master and slave synchronization ................................... 749
   A.8.20 Two timers synchronized by an external trigger ................ 750
   A.8.21 DMA burst feature ..................................................... 751
A.9 IRTIM code examples ........................................................ 752
   A.9.1 TIM16 and TIM17 configuration .................................... 752
   A.9.2 IRQHandler for IRTIM .................................................. 753
A.10 DBG code examples .......................................................... 754
   A.10.1 DBG read device ID .................................................... 754
   A.10.2 DBG debug in Low-power mode ................................... 754
A.11 I2C code examples .......................................................... 754
   A.11.1 I2C configured in master mode to receive ......................... 754
   A.11.2 I2C configured in master mode to transmit ....................... 754
   A.11.3 I2C configured in slave mode ........................................ 755
   A.11.4 I2C master transmitter ................................................. 755
   A.11.5 I2C master receiver ..................................................... 755
   A.11.6 I2C slave transmitter .................................................. 756
   A.11.7 I2C slave receiver ..................................................... 756
A.11.8 I2C configured in master mode to transmit with DMA .......................... 756
A.11.9 I2C configured in slave mode to receive with DMA .......................... 757
A.12  IWDG code examples ................................................................. 757
A.12.1 IWDG configuration ............................................................... 757
A.12.2 IWDG configuration with window ............................................. 758
A.13  RTC code examples ................................................................. 758
A.13.1 RTC calendar configuration .................................................... 758
A.13.2 RTC alarm configuration ........................................................ 759
A.13.3 RTC WUT configuration .......................................................... 759
A.13.4 RTC read calendar ............................................................... 759
A.13.5 RTC calibration ................................................................. 760
A.13.6 RTC tamper and time stamp configuration ................................. 760
A.13.7 RTC tamper and time stamp .................................................. 761
A.13.8 RTC clock output ............................................................... 761
A.14  SPI code examples ................................................................. 761
A.14.1 SPI master configuration ........................................................ 761
A.14.2 SPI slave configuration ........................................................ 762
A.14.3 SPI full duplex communication ................................................. 762
A.14.4 SPI interrupt ................................................................. 762
A.14.5 SPI master configuration with DMA ......................................... 762
A.14.6 SPI slave configuration with DMA ........................................... 763
A.15  USART code examples ............................................................ 763
A.15.1 USART transmitter configuration .............................................. 763
A.15.2 USART transmit byte .......................................................... 763
A.15.3 USART transfer complete ..................................................... 763
A.15.4 USART receiver configuration ............................................... 764
A.15.5 USART receive byte ........................................................... 764
A.15.6 USART synchronous mode .................................................... 764
A.15.7 USART DMA ................................................................. 765
A.15.8 USART hardware flow control ................................................. 765
A.16  WWDG code examples ............................................................ 765
A.16.1 WWDG configuration .......................................................... 765

Important security notice ................................................................. 766

Revision history ................................................................. 767
List of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>STM32F0x0 memory boundary addresses</td>
<td>39</td>
</tr>
<tr>
<td>Table 2</td>
<td>STM32F0x0 peripheral register boundary addresses</td>
<td>40</td>
</tr>
<tr>
<td>Table 3</td>
<td>Boot modes</td>
<td>44</td>
</tr>
<tr>
<td>Table 4</td>
<td>Flash memory organization (STM32F030x4, STM32F030x6, STM32F070x6 and STM32F030x8 devices)</td>
<td>47</td>
</tr>
<tr>
<td>Table 5</td>
<td>Flash memory organization (STM32F070xB, STM32F030xC devices)</td>
<td>48</td>
</tr>
<tr>
<td>Table 6</td>
<td>Flash memory read protection status</td>
<td>56</td>
</tr>
<tr>
<td>Table 7</td>
<td>Access status versus protection level and execution modes</td>
<td>57</td>
</tr>
<tr>
<td>Table 8</td>
<td>Flash interrupt request</td>
<td>58</td>
</tr>
<tr>
<td>Table 9</td>
<td>Flash interface - Register map and reset values</td>
<td>65</td>
</tr>
<tr>
<td>Table 10</td>
<td>Option byte format</td>
<td>66</td>
</tr>
<tr>
<td>Table 11</td>
<td>Option byte organization</td>
<td>66</td>
</tr>
<tr>
<td>Table 12</td>
<td>Option byte map and ST production values</td>
<td>69</td>
</tr>
<tr>
<td>Table 13</td>
<td>CRC internal input/output signals</td>
<td>71</td>
</tr>
<tr>
<td>Table 14</td>
<td>CRC register map and reset values</td>
<td>75</td>
</tr>
<tr>
<td>Table 15</td>
<td>Low-power mode summary</td>
<td>79</td>
</tr>
<tr>
<td>Table 16</td>
<td>Sleep-now</td>
<td>81</td>
</tr>
<tr>
<td>Table 17</td>
<td>Sleep-on-exit</td>
<td>81</td>
</tr>
<tr>
<td>Table 18</td>
<td>Stop mode</td>
<td>82</td>
</tr>
<tr>
<td>Table 19</td>
<td>Standby mode</td>
<td>83</td>
</tr>
<tr>
<td>Table 20</td>
<td>PWR register map and reset values</td>
<td>87</td>
</tr>
<tr>
<td>Table 21</td>
<td>RCC register map and reset values</td>
<td>123</td>
</tr>
<tr>
<td>Table 22</td>
<td>Port bit configuration table</td>
<td>126</td>
</tr>
<tr>
<td>Table 23</td>
<td>GPIO register map and reset values</td>
<td>140</td>
</tr>
<tr>
<td>Table 24</td>
<td>SYSCFG register map and reset values</td>
<td>148</td>
</tr>
<tr>
<td>Table 25</td>
<td>DMA implementation</td>
<td>150</td>
</tr>
<tr>
<td>Table 26</td>
<td>DMA requests for each channel on STM32F030x4/x6/x8 and STM32F070x6/B devices</td>
<td>151</td>
</tr>
<tr>
<td>Table 27</td>
<td>DMA requests for each channel on STM32F030xC devices</td>
<td>152</td>
</tr>
<tr>
<td>Table 28</td>
<td>Programmable data width and endian behavior (when PINC = MINC = 1)</td>
<td>159</td>
</tr>
<tr>
<td>Table 29</td>
<td>DMA interrupt requests</td>
<td>161</td>
</tr>
<tr>
<td>Table 30</td>
<td>DMA register map and reset values</td>
<td>169</td>
</tr>
<tr>
<td>Table 31</td>
<td>Vector table</td>
<td>171</td>
</tr>
<tr>
<td>Table 32</td>
<td>External interrupt/event controller register map and reset values</td>
<td>181</td>
</tr>
<tr>
<td>Table 33</td>
<td>ADC input/output pins</td>
<td>184</td>
</tr>
<tr>
<td>Table 34</td>
<td>ADC internal input/output signals</td>
<td>185</td>
</tr>
<tr>
<td>Table 35</td>
<td>External triggers</td>
<td>185</td>
</tr>
<tr>
<td>Table 36</td>
<td>Latency between trigger and start of conversion</td>
<td>189</td>
</tr>
<tr>
<td>Table 37</td>
<td>Configuring the trigger polarity</td>
<td>193</td>
</tr>
<tr>
<td>Table 38</td>
<td>tSAR timings depending on resolution</td>
<td>195</td>
</tr>
<tr>
<td>Table 39</td>
<td>Analog watchdog comparison</td>
<td>204</td>
</tr>
<tr>
<td>Table 40</td>
<td>Analog watchdog channel selection</td>
<td>204</td>
</tr>
<tr>
<td>Table 41</td>
<td>ADC interrupts</td>
<td>210</td>
</tr>
<tr>
<td>Table 42</td>
<td>ADC register map and reset values</td>
<td>223</td>
</tr>
<tr>
<td>Table 43</td>
<td>Counting direction versus encoder signals</td>
<td>264</td>
</tr>
<tr>
<td>Table 44</td>
<td>TIMx Internal trigger connection</td>
<td>277</td>
</tr>
<tr>
<td>Table 45</td>
<td>Output control bits for complementary OCx and OCxN channels with break feature</td>
<td>290</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>46</td>
<td>TIM1 register map and reset values</td>
<td>299</td>
</tr>
<tr>
<td>47</td>
<td>Counting direction versus encoder signals</td>
<td>331</td>
</tr>
<tr>
<td>48</td>
<td>TIM3 internal trigger connection</td>
<td>348</td>
</tr>
<tr>
<td>49</td>
<td>Output control bit for standard OCx channels</td>
<td>358</td>
</tr>
<tr>
<td>50</td>
<td>TIM3 register map and reset values</td>
<td>365</td>
</tr>
<tr>
<td>51</td>
<td>TIM6/TIM7 register map and reset values</td>
<td>379</td>
</tr>
<tr>
<td>52</td>
<td>Output control bit for standard OCx channels</td>
<td>397</td>
</tr>
<tr>
<td>53</td>
<td>TIM14 register map and reset values</td>
<td>399</td>
</tr>
<tr>
<td>54</td>
<td>TIMx Internal trigger connection</td>
<td>433</td>
</tr>
<tr>
<td>55</td>
<td>Output control bits for complementary OCx and OCxN channels with break feature</td>
<td>443</td>
</tr>
<tr>
<td>56</td>
<td>TIM15 register map and reset values</td>
<td>449</td>
</tr>
<tr>
<td>57</td>
<td>Output control bits for complementary OCx and OCxN channels with break feature</td>
<td>460</td>
</tr>
<tr>
<td>58</td>
<td>TIM16/TIM17 register map and reset values</td>
<td>466</td>
</tr>
<tr>
<td>59</td>
<td>IWDG register map and reset values</td>
<td>477</td>
</tr>
<tr>
<td>60</td>
<td>WWDG register map and reset values</td>
<td>483</td>
</tr>
<tr>
<td>61</td>
<td>STM32F0x0 RTC implementation</td>
<td>485</td>
</tr>
<tr>
<td>62</td>
<td>RTC pin PC13 configuration</td>
<td>488</td>
</tr>
<tr>
<td>63</td>
<td>LSE pin PC14 configuration</td>
<td>489</td>
</tr>
<tr>
<td>64</td>
<td>LSE pin PC15 configuration</td>
<td>489</td>
</tr>
<tr>
<td>65</td>
<td>Effect of low-power modes on RTC</td>
<td>501</td>
</tr>
<tr>
<td>66</td>
<td>Interrupt control bits</td>
<td>501</td>
</tr>
<tr>
<td>67</td>
<td>RTC register map and reset values</td>
<td>521</td>
</tr>
<tr>
<td>68</td>
<td>STM32F0x0 I2C implementation</td>
<td>525</td>
</tr>
<tr>
<td>69</td>
<td>I2C input/output pins</td>
<td>527</td>
</tr>
<tr>
<td>70</td>
<td>I2C internal input/output signals</td>
<td>527</td>
</tr>
<tr>
<td>71</td>
<td>Comparison of analog vs. digital filters</td>
<td>529</td>
</tr>
<tr>
<td>72</td>
<td>I2C-SMBus specification data setup and hold times</td>
<td>532</td>
</tr>
<tr>
<td>73</td>
<td>I2C configuration</td>
<td>536</td>
</tr>
<tr>
<td>74</td>
<td>I2C-SMBus specification clock timings</td>
<td>547</td>
</tr>
<tr>
<td>75</td>
<td>Examples of timing settings for fI2CCLK = 8 MHz</td>
<td>557</td>
</tr>
<tr>
<td>76</td>
<td>Examples of timing settings for fI2CCLK = 16 MHz</td>
<td>557</td>
</tr>
<tr>
<td>77</td>
<td>Examples of timing settings for fI2CCLK = 48 MHz</td>
<td>558</td>
</tr>
<tr>
<td>78</td>
<td>SMBus timeout specifications</td>
<td>560</td>
</tr>
<tr>
<td>79</td>
<td>SMBus with PEC configuration</td>
<td>561</td>
</tr>
<tr>
<td>80</td>
<td>Examples of TIMEOUTA settings (max (t_{\text{TIMEOUT}} = 25) ms)</td>
<td>563</td>
</tr>
<tr>
<td>81</td>
<td>Examples of TIMEOUB settings</td>
<td>563</td>
</tr>
<tr>
<td>82</td>
<td>Examples of TIMEOUTA settings (max (t_{\text{IDLE}} = 50) µs)</td>
<td>563</td>
</tr>
<tr>
<td>83</td>
<td>Effect of low-power modes on the I2C</td>
<td>573</td>
</tr>
<tr>
<td>84</td>
<td>I2C interrupt requests</td>
<td>574</td>
</tr>
<tr>
<td>85</td>
<td>I2C register map and reset values</td>
<td>588</td>
</tr>
<tr>
<td>86</td>
<td>STM32F0x0 USART features</td>
<td>592</td>
</tr>
<tr>
<td>87</td>
<td>Noise detection from sampled data</td>
<td>603</td>
</tr>
<tr>
<td>88</td>
<td>Error calculation for programmed baud rates at (f_{\text{CK}} = 48) MHz in both cases of oversampling by 16 or by 8</td>
<td>605</td>
</tr>
<tr>
<td>89</td>
<td>Tolerance of the USART receiver when BRR [3:0] = 0000</td>
<td>606</td>
</tr>
<tr>
<td>90</td>
<td>Tolerance of the USART receiver when BRR [3:0] is different from 0000</td>
<td>606</td>
</tr>
<tr>
<td>91</td>
<td>Frame formats</td>
<td>610</td>
</tr>
<tr>
<td>92</td>
<td>Effect of low-power modes on the USART</td>
<td>618</td>
</tr>
<tr>
<td>93</td>
<td>USART interrupt requests</td>
<td>619</td>
</tr>
<tr>
<td>94</td>
<td>USART register map and reset values</td>
<td>635</td>
</tr>
<tr>
<td>95</td>
<td>STM32F0x0 SPI implementation</td>
<td>638</td>
</tr>
<tr>
<td>96</td>
<td>SPI interrupt requests</td>
<td>662</td>
</tr>
</tbody>
</table>
Table 97. SPI register map and reset values ................................................................. 671
Table 98. STM32F0x0 USB implementation ................................................................. 672
Table 99. Double-buffering buffer flag definition ....................................................... 682
Table 100. Bulk double-buffering memory buffers usage ........................................... 682
Table 101. Isochronous memory buffers usage ............................................................ 684
Table 102. Resume event detection ............................................................................. 685
Table 103. Reception status encoding ........................................................................ 698
Table 104. Endpoint type encoding ........................................................................... 698
Table 105. Endpoint kind meaning ............................................................................. 698
Table 106. Transmission status encoding ................................................................... 699
Table 107. Definition of allocated buffer memory ....................................................... 702
Table 108. USB register map and reset values ............................................................. 703
Table 109. SW debug port pins .................................................................................. 707
Table 110. DEV_ID and REV_ID field values ............................................................. 708
Table 111. Packet request (8-bits) ............................................................................. 709
Table 112. ACK response (3 bits) ............................................................................. 709
Table 113. DATA transfer (33 bits) ............................................................................ 710
Table 114. SW-DP registers ...................................................................................... 711
Table 115. 32-bit debug port registers addressed through the shifted value A[3:2] .................................................................................. 712
Table 116. Core debug registers ................................................................................. 712
Table 117. DBG register map and reset values ............................................................ 719
Table 118. Document revision history ...................................................................... 767
List of figures

Figure 1. System architecture .................................................. 35
Figure 2. Memory map ............................................................. 38
Figure 3. Programming procedure ............................................. 51
Figure 4. Flash memory Page erase procedure ........................... 53
Figure 5. Flash memory mass erase procedure ......................... 54
Figure 6. CRC calculation unit block diagram ......................... 71
Figure 7. Power supply overview .............................................. 76
Figure 8. Power on reset/power down reset waveform ................ 77
Figure 9. Simplified diagram of the reset circuit ....................... 89
Figure 10. Clock tree (STM32F030x4, STM32F030x6 and STM32F030x8 devices) ................................................. 91
Figure 11. Clock tree (STM32F070x6, STM32F070xB and STM32F030xC) ................................................................. 92
Figure 12. HSE/LSE clock sources ............................................ 93
Figure 13. Frequency measurement with TIM14 in capture mode .... 98
Figure 14. Basic structure of an I/O port bit .............................. 126
Figure 15. Input floating / pull up / pull down configurations ....... 130
Figure 16. Output configuration ............................................... 131
Figure 17. Alternate function configuration ............................... 132
Figure 18. High impedance-analog configuration ....................... 133
Figure 19. DMA request mapping ............................................. 151
Figure 20. DMA block diagram ............................................... 153
Figure 21. Extended interrupts and events controller (EXTI) block diagram ................................................................. 174
Figure 22. External interrupt/event GPIO mapping ..................... 176
Figure 23. ADC block diagram ................................................ 184
Figure 24. ADC calibration ..................................................... 186
Figure 25. Enabling/disabling the ADC ..................................... 187
Figure 26. ADC clock scheme ................................................ 188
Figure 27. Analog to digital conversion time ............................. 192
Figure 28. ADC conversion timings .......................................... 192
Figure 29. Stopping an ongoing conversion ............................... 193
Figure 30. Single conversions of a sequence, software trigger ........ 196
Figure 31. Continuous conversion of a sequence, software trigger .... 196
Figure 32. Single conversions of a sequence, hardware trigger ....... 197
Figure 33. Continuous conversions of a sequence, hardware trigger .... 197
Figure 34. Data alignment and resolution ................................ 198
Figure 35. Example of overrun (OVR) ...................................... 199
Figure 36. Wait mode conversion (continuous mode, software trigger) .......................... 201
Figure 37. Behavior with WAIT = 0, AUTOFF = 1 ....................... 202
Figure 38. Behavior with WAIT = 1, AUTOFF = 1 ....................... 203
Figure 39. Analog watchdog guarded area ................................ 204
Figure 40. ADC_AWD1_OUT signal generation ........................... 205
Figure 41. ADC_AWD1_OUT signal generation (AWD flag not cleared by software) ................... 206
Figure 42. ADC1_AWD_OUT signal generation (on a single channel) ................................................................. 206
Figure 43. Analog watchdog threshold update ............................ 207
Figure 44. Temperature sensor and VREFINT channel block diagram ................................................................. 208
Figure 45. Advanced-control timer block diagram ...................... 226
Figure 46. Counter timing diagram with prescaler division change from 1 to 2 ................... 228
Figure 47. Counter timing diagram with prescaler division change from 1 to 4 ................... 228
Figure 48. Counter timing diagram, internal clock divided by 1 ........ 230
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>Counter timing diagram, internal clock divided by 2</td>
<td>230</td>
</tr>
<tr>
<td>50</td>
<td>Counter timing diagram, internal clock divided by 4</td>
<td>231</td>
</tr>
<tr>
<td>51</td>
<td>Counter timing diagram, internal clock divided by N</td>
<td>231</td>
</tr>
<tr>
<td>52</td>
<td>Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded)</td>
<td>232</td>
</tr>
<tr>
<td>53</td>
<td>Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)</td>
<td>232</td>
</tr>
<tr>
<td>54</td>
<td>Counter timing diagram, internal clock divided by 1</td>
<td>233</td>
</tr>
<tr>
<td>55</td>
<td>Counter timing diagram, internal clock divided by 2</td>
<td>234</td>
</tr>
<tr>
<td>56</td>
<td>Counter timing diagram, internal clock divided by 4</td>
<td>234</td>
</tr>
<tr>
<td>57</td>
<td>Counter timing diagram, internal clock divided by N</td>
<td>234</td>
</tr>
<tr>
<td>58</td>
<td>Counter timing diagram, update event when repetition counter is not used</td>
<td>235</td>
</tr>
<tr>
<td>59</td>
<td>Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6</td>
<td>236</td>
</tr>
<tr>
<td>60</td>
<td>Counter timing diagram, internal clock divided by 2</td>
<td>237</td>
</tr>
<tr>
<td>61</td>
<td>Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36</td>
<td>237</td>
</tr>
<tr>
<td>62</td>
<td>Counter timing diagram, internal clock divided by N</td>
<td>238</td>
</tr>
<tr>
<td>63</td>
<td>Counter timing diagram, update event with ARPE=1 (counter underflow)</td>
<td>238</td>
</tr>
<tr>
<td>64</td>
<td>Counter timing diagram, Update event with ARPE=1 (counter overflow)</td>
<td>239</td>
</tr>
<tr>
<td>65</td>
<td>Update rate examples depending on mode and TIMx_RCR register settings</td>
<td>240</td>
</tr>
<tr>
<td>66</td>
<td>Control circuit in normal mode, internal clock divided by 1</td>
<td>241</td>
</tr>
<tr>
<td>67</td>
<td>TI2 external clock connection example.</td>
<td>242</td>
</tr>
<tr>
<td>68</td>
<td>Control circuit in external clock mode 1</td>
<td>243</td>
</tr>
<tr>
<td>69</td>
<td>External trigger input block</td>
<td>243</td>
</tr>
<tr>
<td>70</td>
<td>Control circuit in external clock mode 2</td>
<td>244</td>
</tr>
<tr>
<td>71</td>
<td>Capture/compare channel (example: channel 1 input stage)</td>
<td>245</td>
</tr>
<tr>
<td>72</td>
<td>Capture/compare channel 1 main circuit</td>
<td>245</td>
</tr>
<tr>
<td>73</td>
<td>Output stage of capture/compare channel (channel 1 to 3)</td>
<td>246</td>
</tr>
<tr>
<td>74</td>
<td>Output stage of capture/compare channel (channel 4)</td>
<td>246</td>
</tr>
<tr>
<td>75</td>
<td>PWM input mode timing</td>
<td>248</td>
</tr>
<tr>
<td>76</td>
<td>Output compare mode, toggle on OC1</td>
<td>250</td>
</tr>
<tr>
<td>77</td>
<td>Edge-aligned PWM waveforms (ARR=8)</td>
<td>252</td>
</tr>
<tr>
<td>78</td>
<td>Center-aligned PWM waveforms (ARR=8)</td>
<td>253</td>
</tr>
<tr>
<td>79</td>
<td>Complementary output with dead-time insertion</td>
<td>255</td>
</tr>
<tr>
<td>80</td>
<td>Dead-time waveforms with delay greater than the negative pulse</td>
<td>255</td>
</tr>
<tr>
<td>81</td>
<td>Dead-time waveforms with delay greater than the positive pulse</td>
<td>255</td>
</tr>
<tr>
<td>82</td>
<td>Output behavior in response to a break</td>
<td>258</td>
</tr>
<tr>
<td>83</td>
<td>Clearing TIMx OCxREF</td>
<td>260</td>
</tr>
<tr>
<td>84</td>
<td>6-step generation, COM example (OSSR=1)</td>
<td>261</td>
</tr>
<tr>
<td>85</td>
<td>Example of one pulse mode</td>
<td>262</td>
</tr>
<tr>
<td>86</td>
<td>Example of counter operation in encoder interface mode</td>
<td>265</td>
</tr>
<tr>
<td>87</td>
<td>Example of encoder interface mode with TI1FP1 polarity inverted</td>
<td>265</td>
</tr>
<tr>
<td>88</td>
<td>Example of hall sensor interface</td>
<td>267</td>
</tr>
<tr>
<td>89</td>
<td>Control circuit in reset mode</td>
<td>268</td>
</tr>
<tr>
<td>90</td>
<td>Control circuit in gated mode</td>
<td>269</td>
</tr>
<tr>
<td>91</td>
<td>Control circuit in trigger mode</td>
<td>270</td>
</tr>
<tr>
<td>92</td>
<td>Control circuit in external clock mode 2 + trigger mode</td>
<td>271</td>
</tr>
<tr>
<td>93</td>
<td>General-purpose timer block diagram (TIM3)</td>
<td>302</td>
</tr>
<tr>
<td>94</td>
<td>Counter timing diagram with prescaler division change from 1 to 2</td>
<td>303</td>
</tr>
<tr>
<td>95</td>
<td>Counter timing diagram with prescaler division change from 1 to 4</td>
<td>304</td>
</tr>
<tr>
<td>96</td>
<td>Counter timing diagram, internal clock divided by 1</td>
<td>305</td>
</tr>
<tr>
<td>97</td>
<td>Counter timing diagram, internal clock divided by 2</td>
<td>305</td>
</tr>
<tr>
<td>98</td>
<td>Counter timing diagram, internal clock divided by 4</td>
<td>306</td>
</tr>
</tbody>
</table>
Figure 99. Counter timing diagram, internal clock divided by N. .................................................... 306
Figure 100. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded) .......... 307
Figure 101. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded) ............. 307
Figure 102. Counter timing diagram, internal clock divided by 1 .................................................... 308
Figure 103. Counter timing diagram, internal clock divided by 2 .................................................... 309
Figure 104. Counter timing diagram, internal clock divided by 4 .................................................... 309
Figure 105. Counter timing diagram, internal clock divided by N .................................................... 310
Figure 106. Counter timing diagram, Update event when repetition counter is not used ................. 310
Figure 107. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6 ......................... 312
Figure 108. Counter timing diagram, internal clock divided by 2 .................................................... 312
Figure 109. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 ..................... 313
Figure 110. Counter timing diagram, internal clock divided by N .................................................... 313
Figure 111. Counter timing diagram, Update event with ARPE=1 (counter underflow). ............... 314
Figure 112. Counter timing diagram, Update event with ARPE=1 (counter overflow) ................. 314
Figure 113. Control circuit in normal mode, internal clock divided by 1 ....................................... 315
Figure 114. TI2 external clock connection example. ................................................................. 316
Figure 115. Control circuit in external clock mode 1 ................................................................. 317
Figure 116. External trigger input block ......................................................................................... 317
Figure 117. Control circuit in external clock mode 2 ................................................................. 318
Figure 118. Capture/compare channel (example: channel 1 input stage) ..................................... 319
Figure 119. Capture/compare channel 1 main circuit ................................................................. 319
Figure 120. Output stage of capture/compare channel (channel 1) .............................................. 320
Figure 121. PWM input mode timing ......................................................................................... 322
Figure 122. Output compare mode, toggle on OC1 ................................................................. 324
Figure 123. Edge-aligned PWM waveforms (ARR=8) ............................................................... 325
Figure 124. Center-aligned PWM waveforms (ARR=8) ............................................................ 327
Figure 125. Example of one-pulse mode ....................................................................................... 328
Figure 126. Clearing TIMx OCxREF ......................................................................................... 330
Figure 127. Example of counter operation in encoder interface mode ........................................... 332
Figure 128. Example of encoder interface mode with TI1FP1 polarity inverted ......................... 332
Figure 129. Clearing TIMx OCxREF ......................................................................................... 333
Figure 130. Control circuit in gated mode ..................................................................................... 334
Figure 131. Control circuit in trigger mode .................................................................................... 335
Figure 132. Control circuit in external clock mode 2 + trigger mode ............................................ 336
Figure 133. Master/Slave timer example ..................................................................................... 337
Figure 134. Gating timer 3 with OC1REF of timer 1 ................................................................. 338
Figure 135. Gating timer 3 with Enable of timer 1 ......................................................................... 339
Figure 136. Triggering timer 3 with update of timer 1 ................................................................. 340
Figure 137. Triggering timer 3 with Enable of timer 1 ................................................................. 341
Figure 138. Triggering timer 1 and 3 with timer 1 TI1 input ......................................................... 342
Figure 139. Basic timer block diagram ........................................................................................ 367
Figure 140. Counter timing diagram with prescaler division change from 1 to 2 ......................... 369
Figure 141. Counter timing diagram with prescaler division change from 1 to 4 ......................... 369
Figure 142. Counter timing diagram, internal clock divided by 1 .................................................. 370
Figure 143. Counter timing diagram, internal clock divided by 2 .................................................. 371
Figure 144. Counter timing diagram, internal clock divided by 4 .................................................. 371
Figure 145. Counter timing diagram, internal clock divided by N .................................................. 372
Figure 146. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded) .... 372
Figure 147. Counter timing diagram, update event when ARPE=1
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>148.</td>
<td>Control circuit in normal mode, internal clock divided by 1</td>
<td>374</td>
</tr>
<tr>
<td>149.</td>
<td>General-purpose timer block diagram (TIM14)</td>
<td>381</td>
</tr>
<tr>
<td>150.</td>
<td>Counter timing diagram with prescaler division change from 1 to 2</td>
<td>382</td>
</tr>
<tr>
<td>151.</td>
<td>Counter timing diagram with prescaler division change from 1 to 4</td>
<td>382</td>
</tr>
<tr>
<td>152.</td>
<td>Counter timing diagram, internal clock divided by 1</td>
<td>383</td>
</tr>
<tr>
<td>153.</td>
<td>Counter timing diagram, internal clock divided by 2</td>
<td>383</td>
</tr>
<tr>
<td>154.</td>
<td>Counter timing diagram, internal clock divided by 4</td>
<td>384</td>
</tr>
<tr>
<td>155.</td>
<td>Counter timing diagram, internal clock divided by N</td>
<td>384</td>
</tr>
<tr>
<td>156.</td>
<td>Counter timing diagram, update event when ARPE=0</td>
<td>384</td>
</tr>
<tr>
<td>157.</td>
<td>Counter timing diagram, update event when ARPE=1</td>
<td>385</td>
</tr>
<tr>
<td>158.</td>
<td>Control circuit in normal mode, internal clock divided by 1</td>
<td>385</td>
</tr>
<tr>
<td>159.</td>
<td>Capture/compare channel (example: channel 1 input stage)</td>
<td>386</td>
</tr>
<tr>
<td>160.</td>
<td>Capture/compare channel 1 main circuit</td>
<td>386</td>
</tr>
<tr>
<td>161.</td>
<td>Output stage of capture/compare channel (channel 1)</td>
<td>387</td>
</tr>
<tr>
<td>162.</td>
<td>Output compare mode, toggle on OC1</td>
<td>389</td>
</tr>
<tr>
<td>163.</td>
<td>Edge-aligned PWM waveforms (ARR=8)</td>
<td>391</td>
</tr>
<tr>
<td>164.</td>
<td>TIM15 block diagram</td>
<td>402</td>
</tr>
<tr>
<td>165.</td>
<td>TIM16 and TIM17 block diagram</td>
<td>404</td>
</tr>
<tr>
<td>166.</td>
<td>Counter timing diagram with prescaler division change from 1 to 2</td>
<td>405</td>
</tr>
<tr>
<td>167.</td>
<td>Counter timing diagram with prescaler division change from 1 to 4</td>
<td>406</td>
</tr>
<tr>
<td>168.</td>
<td>Counter timing diagram, internal clock divided by 1</td>
<td>407</td>
</tr>
<tr>
<td>169.</td>
<td>Counter timing diagram, internal clock divided by 2</td>
<td>408</td>
</tr>
<tr>
<td>170.</td>
<td>Counter timing diagram, internal clock divided by 4</td>
<td>408</td>
</tr>
<tr>
<td>171.</td>
<td>Counter timing diagram, internal clock divided by N</td>
<td>409</td>
</tr>
<tr>
<td>172.</td>
<td>Counter timing diagram, update event when ARPE=0</td>
<td>409</td>
</tr>
<tr>
<td>173.</td>
<td>Counter timing diagram, update event when ARPE=1</td>
<td>410</td>
</tr>
<tr>
<td>174.</td>
<td>Update rate examples depending on mode and TIMx_RCR register settings</td>
<td>411</td>
</tr>
<tr>
<td>175.</td>
<td>Control circuit in normal mode, internal clock divided by 1</td>
<td>412</td>
</tr>
<tr>
<td>176.</td>
<td>T12 external clock connection example.</td>
<td>412</td>
</tr>
<tr>
<td>177.</td>
<td>Control circuit in external clock mode 1</td>
<td>413</td>
</tr>
<tr>
<td>178.</td>
<td>Capture/compare channel (example: channel 1 input stage)</td>
<td>414</td>
</tr>
<tr>
<td>179.</td>
<td>Capture/compare channel 1 main circuit</td>
<td>414</td>
</tr>
<tr>
<td>180.</td>
<td>Output stage of capture/compare channel (channel 1)</td>
<td>415</td>
</tr>
<tr>
<td>181.</td>
<td>Output stage of capture/compare channel (channel 2 for TIM15)</td>
<td>415</td>
</tr>
<tr>
<td>182.</td>
<td>PWM input mode timing</td>
<td>417</td>
</tr>
<tr>
<td>183.</td>
<td>Output compare mode, toggle on OC1</td>
<td>419</td>
</tr>
<tr>
<td>184.</td>
<td>Edge-aligned PWM waveforms (ARR=8)</td>
<td>420</td>
</tr>
<tr>
<td>185.</td>
<td>Complementary output with dead-time insertion</td>
<td>421</td>
</tr>
<tr>
<td>186.</td>
<td>Dead-time waveforms with delay greater than the negative pulse</td>
<td>421</td>
</tr>
<tr>
<td>187.</td>
<td>Dead-time waveforms with delay greater than the positive pulse</td>
<td>421</td>
</tr>
<tr>
<td>188.</td>
<td>Output behavior in response to a break</td>
<td>424</td>
</tr>
<tr>
<td>189.</td>
<td>Example of One-pulse mode</td>
<td>425</td>
</tr>
<tr>
<td>190.</td>
<td>Control circuit in reset mode</td>
<td>427</td>
</tr>
<tr>
<td>191.</td>
<td>Control circuit in gated mode</td>
<td>428</td>
</tr>
<tr>
<td>192.</td>
<td>Control circuit in trigger mode</td>
<td>429</td>
</tr>
<tr>
<td>193.</td>
<td>Independent watch dog block diagram</td>
<td>468</td>
</tr>
<tr>
<td>194.</td>
<td>IRTIM internal hardware connections</td>
<td>469</td>
</tr>
</tbody>
</table>

Figure 148. (TIMx_ARR preloaded) .......................................................... 373
Figure 149. Control circuit in gated mode ............................................ 428
Figure 150. Control circuit in gated mode ............................................ 428
Figure 151. (TIMx_ARR not preloaded) ..................................................... 373
Figure 152. Counter timing diagram, update event when ARPE=0 ..................... 409
Figure 153. Counter timing diagram, update event when ARPE=1 ........................ 409
Figure 154. Counter timing diagram, update event when ARPE=0 ........................ 409
Figure 155. Counter timing diagram, update event when ARPE=0 ........................ 409
Figure 156. (TIMx_ARR preloaded) .......................................................... 385
Figure 157. Counter timing diagram, update event when ARPE=1 ........................ 409
Figure 158. Counter timing diagram, update event when ARPE=0 ........................ 409
Figure 159. Capture/compare channel (example: channel 1 input stage) ................ 386
Figure 160. Capture/compare channel 1 main circuit .................................. 386
Figure 161. Output stage of capture/compare channel (channel 1) ........................ 387
Figure 162. Output compare mode, toggle on OC1 ....................................... 390
Figure 163. Edge-aligned PWM waveforms (ARR=8) ....................................... 391
Figure 164. TIM15 block diagram .............................................................. 402
Figure 165. TIM16 and TIM17 block diagram .............................................. 404
Figure 166. Counter timing diagram with prescaler division change from 1 to 2 ........ 405
Figure 167. Counter timing diagram with prescaler division change from 1 to 4 ........ 406
Figure 168. Counter timing diagram, internal clock divided by 1 ...................... 407
Figure 169. Counter timing diagram, internal clock divided by 2 ...................... 408
Figure 170. Counter timing diagram, internal clock divided by 4 ...................... 408
Figure 171. Counter timing diagram, internal clock divided by N ...................... 409
Figure 172. Counter timing diagram, update event when ARPE=0 ........................ 409
Figure 173. Counter timing diagram, update event when ARPE=1 ........................ 410
Figure 174. Update rate examples depending on mode and TIMx_RCR register settings .... 411
Figure 175. Control circuit in normal mode, internal clock divided by 1 .............. 412
Figure 176. T12 external clock connection example ..................................... 412
Figure 177. Control circuit in external clock mode 1 .................................. 413
Figure 178. Capture/compare channel (example: channel 1 input stage) ................ 414
Figure 179. Capture/compare channel 1 main circuit ................................... 414
Figure 180. Output stage of capture/compare channel (channel 1) ........................ 415
Figure 181. Output stage of capture/compare channel (channel 2 for TIM15) ............ 415
Figure 182. PWM input mode timing ............................................................ 417
Figure 183. Output compare mode, toggle on OC1 ....................................... 419
Figure 184. Edge-aligned PWM waveforms (ARR=8) ....................................... 420
Figure 185. Complementary output with dead-time insertion ............................ 421
Figure 186. Dead-time waveforms with delay greater than the negative pulse .......... 421
Figure 187. Dead-time waveforms with delay greater than the positive pulse .......... 421
Figure 188. Output behavior in response to a break .................................... 424
Figure 189. Example of One-pulse mode ..................................................... 425
Figure 190. Control circuit in reset mode ............................................... 427
Figure 191. Control circuit in gated mode .............................................. 428
Figure 192. Control circuit in trigger mode ............................................ 429
Figure 193. IRTIM internal hardware connections ....................................... 468
Figure 194. Independent watch dog block diagram ...................................... 469
Figure 195. Watchdog block diagram .......................................................... 479
Figure 196. Window watchdog timing diagram ............................................. 480
Figure 197. RTC block diagram in STM32F030x4/6,
             STM32F070x6 and STM32F030x8 devices .................................. 486
Figure 198. RTC block diagram for STM32F070xB
             and STM32F030xC devices ..................................................... 487
Figure 199. I2C block diagram ................................................................. 526
Figure 200. I2C2 block diagram ............................................................... 527
Figure 201. I2C bus protocol ................................................................. 529
Figure 202. Setup and hold timings ........................................................... 530
Figure 203. I2C initialization flow ............................................................. 533
Figure 204. Data reception ..................................................................... 534
Figure 205. Data transmission ................................................................. 535
Figure 206. Slave initialization flow .......................................................... 538
Figure 207. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 0  . 540
Figure 208. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 1 . 541
Figure 209. Transfer bus diagrams for I2C slave transmitter (mandatory events only) ................................................................. 542
Figure 210. Transfer sequence flow for slave receiver with NOSTRETCH = 0  . 543
Figure 211. Transfer sequence flow for slave receiver with NOSTRETCH = 1 . 544
Figure 212. Transfer bus diagrams for I2C slave receiver
            (mandatory events only) ......................................................... 544
Figure 213. Master clock generation .......................................................... 546
Figure 214. Master initialization flow ....................................................... 548
Figure 215. 10-bit address read access with HEAD10R = 0 ............................ 548
Figure 216. 10-bit address read access with HEAD10R = 1 ............................ 549
Figure 217. Transfer sequence flow for I2C master transmitter for N ≤ 255 bytes 550
Figure 218. Transfer sequence flow for I2C master transmitter for N > 255 bytes 551
Figure 219. Transfer bus diagrams for I2C master transmitter
            (mandatory events only) ......................................................... 552
Figure 220. Transfer sequence flow for I2C master receiver for N ≤ 255 bytes 554
Figure 221. Transfer sequence flow for I2C master receiver for N > 255 bytes 555
Figure 222. Transfer bus diagrams for I2C master receiver
            (mandatory events only) ......................................................... 556
Figure 223. Timeout intervals for tLOW:SEXT, tLOW:MEXT .......................... 560
Figure 224. Transfer sequence flow for SMBus slave transmitter N bytes + PEC . 564
Figure 225. Transfer bus diagrams for SMBus slave transmitter (SBC = 1) .......................... 564
Figure 226. Transfer sequence flow for SMBus slave receiver N bytes + PEC . 566
Figure 227. Bus transfer diagrams for SMBus slave receiver (SBC = 1) .......................... 567
Figure 228. Bus transfer diagrams for SMBus master transmitter .......................... 568
Figure 229. Bus transfer diagrams for SMBus master receiver .......................... 570
Figure 230. USART block diagram ............................................................ 594
Figure 231. Word length programming ....................................................... 595
Figure 232. Configurable stop bits ............................................................ 596
Figure 233. TC/TXE behavior when transmitting .......................................... 598
Figure 234. Start bit detection when oversampling by 16 or 8 .......................... 599
Figure 235. Data sampling when oversampling by 16 ..................................... 602
Figure 236. Data sampling when oversampling by 8 ..................................... 603
Figure 237. Mute mode using idle line detection ......................................... 608
Figure 238. Mute mode using address mark detection ................................... 609
Figure 239. USART example of synchronous transmission. .......................... 611
Figure 240. USART data clock timing diagram (M=0) .................................. 612
Figure 241. USART data clock timing diagram (M=1) .................................. 612
Figure 242. RX data setup/hold time ................................................................. 613
Figure 243. Transmission using DMA ............................................................... 615
Figure 244. Reception using DMA ...................................................................... 616
Figure 245. Hardware flow control between 2 USARTs ....................................... 616
Figure 246. RS232 RTS flow control ................................................................. 617
Figure 247. RS232 CTS flow control .................................................................. 618
Figure 248. USART interrupt mapping diagram .................................................. 619
Figure 249. SPI block diagram ......................................................................... 639
Figure 250. Full-duplex single master/ single slave application ............................ 640
Figure 251. Half-duplex single master/ single slave application ........................... 640
Figure 252. Simplex single master/single slave application (master in transmit-only/
slave in receive-only mode) ............................................................................ 641
Figure 253. Master and three independent slaves. ............................................... 642
Figure 254. Multimaster application .................................................................. 643
Figure 255. Hardware/software slave select management .................................... 644
Figure 256. Data clock timing diagram ................................................................ 645
Figure 257. Data alignment when data length is not equal to 8-bit or 16-bit .......... 646
Figure 258. Packing data in FIFO for transmission and reception ....................... 650
Figure 259. Master full-duplex communication .................................................. 653
Figure 260. Slave full-duplex communication ..................................................... 654
Figure 261. Master full-duplex communication with CRC .................................... 655
Figure 262. Master full-duplex communication in packed mode ......................... 656
Figure 263. NSSP pulse generation in Motorola SPI master mode ...................... 659
Figure 264. TI mode transfer ............................................................................ 660
Figure 265. USB peripheral block diagram ....................................................... 673
Figure 266. Packet buffer areas with examples of buffer description table locations 677
Figure 267. Block diagram of STM32F0x0 MCU and Arm® Cortex®-M0-level debug support 705
1 Documentation conventions

1.1 General information
The STM32F0x0 devices have an Arm® Cortex® -M0 core.

1.2 List of abbreviations for registers
The following abbreviations(b) are used in register descriptions:

- **read/write (rw)** Software can read and write to this bit.
- **read-only (r)** Software can only read this bit.
- **write-only (w)** Software can only write to this bit. Reading this bit returns the reset value.
- **read/clear write0 (rc_w0)** Software can read as well as clear this bit by writing 0. Writing 1 has no effect on the bit value.
- **read/clear write1 (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 write (rc_w)** Software can read as well as clear this bit by writing to the register. The value written to this bit is not important.
- **read/clear by read (rc_r)** Software can read this bit. Reading this bit automatically clears it to 0. Writing this bit has no effect on the bit value.
- **read/set by read (rs_r)** Software can read this bit. Reading this bit automatically sets it to 1. Writing this bit has no effect on the bit value.
- **read/set (rs)** Software can read as well as set this bit. Writing 0 has no effect on the bit value.
- **read/write once (rwo)** Software can only write once to this bit and can also read it at any time. Only a reset can return the bit to its reset value.
- **toggle (t)** The software can toggle this bit by writing 1. Writing 0 has no effect.
- **read-only write trigger (rt_w1)** Software can read this bit. Writing 1 triggers an event but has no effect on the bit value.
- **Reserved (Res.)** Reserved bit, must be kept at reset value.

---

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
b. This is an exhaustive list of all abbreviations applicable to STMicroelectronics microcontrollers, some of them may not be used in the current document.
1.3 Glossary

This section gives a brief definition of acronyms and abbreviations used in this document:

- **Word**: data of 32-bit length.
- **Half-word**: data of 16-bit length.
- **Byte**: data of 8-bit length.
- **SWD-DP (SWD DEBUG PORT)**: SWD-DP provides a 2-pin (clock and data) interface based on the Serial Wire Debug (SWD) protocol. Please refer to the Arm® Cortex®-M0 technical reference manual.
- **IAP (in-application programming)**: IAP is the ability to re-program the flash memory of a microcontroller while the user program is running.
- **ICP (in-circuit programming)**: ICP is the ability to program the flash memory of a microcontroller using the JTAG protocol, the SWD protocol or the bootloader while the device is mounted on the user application board.
- **Option bytes**: product configuration bits stored in the flash memory.
- **OBL**: option byte loader.
- **AHB**: advanced high-performance bus.
- **APB**: advanced peripheral bus.

1.4 Availability of peripherals

For availability of peripherals and their number across all sales types, refer to the particular device datasheet.
2 System and memory overview

2.1 System architecture

The main system consists of:
- Two masters:
  - Arm® Cortex®-M0 core
  - General-purpose DMA
- Four slaves:
  - Internal SRAM
  - Internal flash memory
  - AHB1 with AHB to APB bridge, which connects all the APB peripherals
  - AHB2 dedicated to GPIO ports

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

**Figure 1. System architecture**

- ARM® Cortex®-M0 core
- DMA1 channels 1 to 5
- System bus
- Bus matrix
- Flash memory
- FLITF Flash interface
- SRAM
- GPIO Ports A,B,C,D,F
- AHB2 bus
- AHB1 bus
- AHB to APB bridge
- APB bus
- Reset and clock controller (RCC)
- CRC
- SYSCFG, ADC, TIM1, TIM3, TIM6, TIM7, TIM14 to TIM17, IWDG, WWDG, RTC, PWR, I2C1, I2C2, USART1 to USART6, SPI1, SPI2, DBGMCU
- DMA requests

**System bus**

This bus connects the system bus of the Arm® Cortex®-M0 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 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 two masters (CPU, DMA) and four slaves (FLITF, SRAM, AHB1 with AHB to APB bridge and AHB2).

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

AHB to APB bridge (APB)
The AHB to APB bridge provides full synchronous connections between the AHB and the APB bus.

Refer to Section 2.2.2: Memory map and register boundary addresses for the address mapping of the peripherals connected to this bridge.

After each device reset, all peripheral clocks are disabled (except for the SRAM and flash). 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

2.2.1 Introduction

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.

The addressable memory space is divided into eight main blocks, of 512 Mbytes each.
2.2.2 Memory map and register boundary addresses

All the memory map areas not allocated to on-chip memories and peripherals are considered “Reserved”. For the detailed mapping of available memory and register areas, refer to the following table, which gives the boundary addresses of the available peripherals.
<table>
<thead>
<tr>
<th>Device</th>
<th>Boundary address</th>
<th>Size</th>
<th>Memory Area</th>
<th>Register description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F030x4,</td>
<td>0x2000 1000 - 0x3FFF FFFF</td>
<td>~512 MB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>STM32F030x6</td>
<td>0x2000 0000 - 0x2000 0FFF</td>
<td>4 KB</td>
<td>SRAM</td>
<td>Section 2.3 on page 42</td>
</tr>
<tr>
<td></td>
<td>0x1FFF FC00 - 0x1FFF FFFF</td>
<td>1 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x1FFF F800 - 0x1FFF FBFF</td>
<td>1 KB</td>
<td>Option bytes</td>
<td>Section 4 on page 66</td>
</tr>
<tr>
<td></td>
<td>0x1FFF EC00 - 0x1FFF F7FF</td>
<td>3 KB</td>
<td>System memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 8000 - 0x1FFF EBFF</td>
<td>~384 MB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 0000 - 0x0800 7FFF</td>
<td>32 KB(1)</td>
<td>Main flash memory</td>
<td>Section 3 on page 46</td>
</tr>
<tr>
<td></td>
<td>0x0000 8000 - 0x07FF FFFF</td>
<td>~128 MB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0000 0000 - 0x0000 7FFF</td>
<td>32 KB(1)</td>
<td>Main flash memory, system memory or SRAM depending on BOOT configuration</td>
<td></td>
</tr>
</tbody>
</table>
**Table 1. STM32F0x0 memory boundary addresses (continued)**

<table>
<thead>
<tr>
<th>Device</th>
<th>Boundary address</th>
<th>Size</th>
<th>Memory Area</th>
<th>Register description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F070xB</td>
<td>0x2000 4000 - 0x3FFF FFFF</td>
<td>~512 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x2000 0000 - 0x2000 3FFF</td>
<td>16 KB</td>
<td>SRAM</td>
<td>Section 2.3 on page 42</td>
</tr>
<tr>
<td></td>
<td>0x11FF F800 - 0x11FF FFFF</td>
<td>2 KB</td>
<td>Option bytes</td>
<td>Section 4 on page 66</td>
</tr>
<tr>
<td></td>
<td>0x0802 0000 - 0x11FF C7FF</td>
<td>~384 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x0800 0000 - 0x0801 FFFF</td>
<td>128 KB</td>
<td>Main flash memory</td>
<td>Section 3 on page 46</td>
</tr>
<tr>
<td></td>
<td>0x0002 0000 - 0x07FF FFFF</td>
<td>~128 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x0000 0000 - 0x0001 FFFF</td>
<td>128 KB</td>
<td>Main flash memory, system memory or SRAM depending on BOOT configuration</td>
<td>-</td>
</tr>
<tr>
<td>STM32F030xC</td>
<td>0x2000 8000 - 0x3FFF FFFF</td>
<td>~512 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x2000 0000 - 0x2000 7FFF</td>
<td>32 KB</td>
<td>SRAM</td>
<td>Section 2.3 on page 42</td>
</tr>
<tr>
<td></td>
<td>0x11FF F800 - 0x11FF FFFF</td>
<td>2 KB</td>
<td>Option bytes</td>
<td>Section 4 on page 66</td>
</tr>
<tr>
<td></td>
<td>0x0804 0000 - 0x11FF D7FF</td>
<td>~384 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x0800 0000 - 0x0803 FFFF</td>
<td>256 KB</td>
<td>Main flash memory</td>
<td>Section 3 on page 46</td>
</tr>
<tr>
<td></td>
<td>0x0004 0000 - 0x07FF FFFF</td>
<td>~128 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x0000 0000 - 0x0003 FFFF</td>
<td>256 KB</td>
<td>Main flash memory, system memory or SRAM depending on BOOT configuration</td>
<td>-</td>
</tr>
</tbody>
</table>

1. Limited to 16 KB on STM32F030x4 devices.

**Table 2. STM32F0x0 peripheral register boundary addresses**

<table>
<thead>
<tr>
<th>Bus</th>
<th>Boundary address</th>
<th>Size</th>
<th>Peripheral</th>
<th>Peripheral register map</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xE000 0000 - 0xE00F FFFF</td>
<td>1MB</td>
<td>Cortex®-M0 internal peripherals</td>
<td>-</td>
</tr>
<tr>
<td>AHB2</td>
<td>0x4800 1400 - 0x4800 17FF</td>
<td>1KB</td>
<td>GPIOF</td>
<td>Section 8.4.11 on page 139</td>
</tr>
<tr>
<td></td>
<td>0x4800 1000 - 0x4800 13FF</td>
<td>1KB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0x4800 0C00 - 0x4800 0FFF</td>
<td>1KB</td>
<td>GPIOD</td>
<td>Section 8.4.11 on page 139</td>
</tr>
<tr>
<td></td>
<td>0x4800 0800 - 0x4800 0BFF</td>
<td>1KB</td>
<td>GPIOC</td>
<td>Section 8.4.11 on page 139</td>
</tr>
<tr>
<td></td>
<td>0x4800 0400 - 0x4800 07FF</td>
<td>1KB</td>
<td>GPIOB</td>
<td>Section 8.4.11 on page 139</td>
</tr>
<tr>
<td></td>
<td>0x4800 0000 - 0x4800 03FF</td>
<td>1KB</td>
<td>GPIOA</td>
<td>Section 8.4.11 on page 139</td>
</tr>
<tr>
<td></td>
<td>0x4002 4400 - 0x47FF FFFF</td>
<td>~128 MB</td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td>Bus</td>
<td>Boundary address</td>
<td>Size</td>
<td>Peripheral</td>
<td>Peripheral register map</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
<td>------</td>
<td>------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>AHB1</td>
<td>0x4002 3400 - 0x4002 43FF</td>
<td>4 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4002 3000 - 0x4002 33FF</td>
<td>1 KB</td>
<td>CRC</td>
<td>Section 5.4.5 on page 75</td>
</tr>
<tr>
<td></td>
<td>0x4002 2400 - 0x4002 2FFF</td>
<td>3 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4002 2000 - 0x4002 23FF</td>
<td>1 KB</td>
<td>FLASH interface</td>
<td>Section 3.5.9 on page 65</td>
</tr>
<tr>
<td></td>
<td>0x4002 1400 - 0x4002 1FFF</td>
<td>3 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4002 1000 - 0x4002 13FF</td>
<td>1 KB</td>
<td>RCC</td>
<td>Section 7.4.15 on page 123</td>
</tr>
<tr>
<td></td>
<td>0x4002 0400 - 0x4002 0FFF</td>
<td>3 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4002 0000 - 0x4002 03FF</td>
<td>1 KB</td>
<td>DMA</td>
<td>Section 10.6.8 on page 169</td>
</tr>
<tr>
<td></td>
<td>0x4001 8000 - 0x4001 FFFF</td>
<td>32 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>APB</td>
<td>0x4001 5C00 - 0x4001 7FFF</td>
<td>9 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 5800 - 0x4001 5BFF</td>
<td>1 KB</td>
<td>DBGMCU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 4C00 - 0x4001 57FF</td>
<td>3 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 4800 - 0x4001 4BFF</td>
<td>1 KB</td>
<td>TIM17</td>
<td>Section 17.6.17 on page 466</td>
</tr>
<tr>
<td></td>
<td>0x4001 4400 - 0x4001 47FF</td>
<td>1 KB</td>
<td>TIM16</td>
<td>Section 17.6.17 on page 466</td>
</tr>
<tr>
<td></td>
<td>0x4001 4000 - 0x4001 43FF</td>
<td>1 KB</td>
<td>TIM15</td>
<td>Section 17.5.19 on page 449</td>
</tr>
<tr>
<td></td>
<td>0x4001 3C00 - 0x4001 3FFF</td>
<td>1 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 3800 - 0x4001 3BFF</td>
<td>1 KB</td>
<td>USART1</td>
<td>Section 23.7.11 on page 635</td>
</tr>
<tr>
<td></td>
<td>0x4001 3400 - 0x4001 37FF</td>
<td>1 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 3000 - 0x4001 33FF</td>
<td>1 KB</td>
<td>SPI1</td>
<td>Section 24.6.8 on page 671</td>
</tr>
<tr>
<td></td>
<td>0x4001 2C00 - 0x4001 2FFF</td>
<td>1 KB</td>
<td>TIM1</td>
<td>Section 13.4.21 on page 299</td>
</tr>
<tr>
<td></td>
<td>0x4001 2800 - 0x4001 27FF</td>
<td>1 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 2400 - 0x4001 23FF</td>
<td>1 KB</td>
<td>ADC</td>
<td>Section 12.11 on page 223</td>
</tr>
<tr>
<td></td>
<td>0x4001 1800 - 0x4001 17FF</td>
<td>4 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 1400 - 0x4001 13FF</td>
<td>1 KB</td>
<td>USART6</td>
<td>Section 23.7.11 on page 635</td>
</tr>
<tr>
<td></td>
<td>0x4001 0800 - 0x4001 07FF</td>
<td>7 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4001 0400 - 0x4001 0FFF</td>
<td>1 KB</td>
<td>EXTI</td>
<td>Section 11.3.7 on page 181</td>
</tr>
<tr>
<td></td>
<td>0x4001 0000 - 0x4001 03FF</td>
<td>1 KB</td>
<td>SYSCFG</td>
<td>Section 9.1.7 on page 148</td>
</tr>
<tr>
<td></td>
<td>0x4000 8000 - 0x4000 FFFF</td>
<td>32 KB</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
STM32F030x4 and STM32F030x6 devices feature 4 Kbytes of SRAM. STM32F030x8 devices feature 8 Kbytes of SRAM. STM32F030xC devices feature 32 Kbytes of SRAM. STM32F070x6 devices feature 6 Kbytes of SRAM. STM32F070xB devices feature 16 Kbytes of SRAM.

The SRAM can be accessed by bytes, half-words (16 bits) or full words (32 bits), at maximum system clock frequency without wait state and thus by both CPU and DMA.

**Parity check**

The user can enable the parity check using the option bit RAM_PARITY_CHECK in the user option byte (refer to Section 4: Option bytes).
The data bus width is 36 bits because 4 bits are available for parity check (1 bit per byte) in order to increase memory robustness, as required for instance by Class B or SIL norms.

The parity bits are computed and stored when writing into the SRAM. Then, they are automatically checked when reading. If one bit fails, an NMI is generated. In addition, to get the SRAM parity error at the same cycle time that it is occurring, a bus error is generated (triggering a HardFault exception) together with the NMI. This avoids the use of corrupted data by the application, but with the side effect of having both NMI and HardFault interrupts generated. The same error can also be linked to the BRK_IN Break input of TIM1/15/16/17, with the SRAM_PARITY_LOCK control bit in the SYSCFG configuration register 2 (SYSCFG_CFGR2). The SRAM Parity Error flag (SRAM_PEF) is available in the SYSCFG configuration register 2 (SYSCFG_CFGR2).

**Note:** When enabling the RAM parity check, it is advised to initialize by software the whole RAM memory at the beginning of the code, to avoid getting parity errors when reading non-initialized locations.

### 2.4 Flash memory overview

The flash memory is composed of two distinct physical areas:

- The main flash memory block. It contains the application program and user data if necessary.
- The information block. It is composed of two parts:
  - option bytes for hardware and memory protection user configuration
  - system memory which contains the proprietary boot loader code (refer to Section 3: Embedded flash memory for more details.)

The flash interface implements instruction access and data access based on the AHB protocol. It implements the prefetch buffer that speeds up CPU code execution. It also implements the logic necessary to carry out the flash memory operations (Program/Erase) controlled through the flash registers.
2.5 Boot configuration

In the STM32F0x0, three different boot modes can be selected through the BOOT0 pin and boot configuration bits nBOOT1 in the User option byte, as shown in the following table.

Table 3. Boot modes

<table>
<thead>
<tr>
<th>nBOOT1 bit</th>
<th>BOOT0 pin</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>Main Flash memory is selected as boot area(^{(1)})</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>System memory is selected as boot area</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Embedded SRAM is selected as boot area</td>
</tr>
</tbody>
</table>

1. For STM32F070x6 and STM32F030xC devices, see also Empty check description.

The boot mode configuration is latched on the 4th rising edge of SYSCLK after a reset. It is up to the user to set boot mode configuration related to the required boot mode.

The boot mode configuration is 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 at 0x0000 0004.

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 (0x0800 0000). In other words, the flash memory contents can be accessed starting from address 0x0000 0000 or 0x0800 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 EC00 on STM32F030x4, STM32F030x6 and STM32F030x8 devices, 0x1FFF C400 on STM32F070x6 devices, 0x1FFF C800 on STM32F070xB and 0x1FFF D800 on STM32F030xC devices).
- Boot from the embedded SRAM: the SRAM is aliased in the boot memory space (0x0000 0000), but it is still accessible from its original memory space (0x2000 0000).

Empty check

On STM32F070x6 and STM32F030xC devices only, internal empty check flag is implemented to allow easy programming of virgin devices by the boot loader. This flag is used when BOOT0 pin is defining Main flash memory as the target boot area. When the flag is set, the device is considered as empty and System memory (boot loader) is selected instead of the main flash as a boot area to allow user to program the flash memory.

Therefore, some of the GPIOs are be reconfigured from the High-Z state. Refer to AN2606 for more details concerning the boot loader and GPIO configuration in System memory boot mode. It is possible to disable this feature by configuring the option bytes, to force boot from the Main flash memory (nSWBOOT0 = 0, nBOOT0 = 1).

The empty check flag is updated only during the loading of option bytes: it is set when the content of the address 0x0800 0000 is read as 0xFFFF FFFF, otherwise it is cleared. It
means a power reset or setting of OBL_LAUNCH bit in FLASH_CR register is needed to clear this flag after programming of a virgin device to execute user code after System reset.

**Note:** *If the device is programmed for a first time but the option bytes are not reloaded, the device still selects System memory as a boot area after a System reset. In the STM32F070x6, the boot loader code is able to detect this situation. It then changes the boot memory mapping to main flash and performs a jump to user code programmed there. In the STM32F030xC, a POR must be performed or the Option bytes reloaded before applying the system reset.*

**Physical remap**

Once the boot mode is selected, the application software can modify the memory accessible in the code area. This modification is performed by programming the MEM_MODE bits in the **SYSCFG configuration register 1 (SYSCFG_CFGR1)**. Unlike Cortex® M3 and M4, the M0 CPU does not support the vector table relocation. For application code which is located in a different address than 0x0800 0000, some additional code must be added in order to be able to serve the application interrupts. A solution is to relocate by software the vector table to the internal SRAM:

- Copy the vector table from the flash (mapped at the base of the application load address) to the base address of the SRAM at 0x2000 0000.
- Remap SRAM at address 0x0000 0000, using SYSCFG configuration register 1.
- Then once an interrupt occurs, the Cortex®-M0 processor fetches the interrupt handler start address from the relocated vector table in SRAM, then it jumps to execute the interrupt handler located in the flash.

This operation should be done at the initialization phase of the application. Please refer to AN4065 and attached IAP code from www.st.com for more details.

**Embedded boot loader**

The embedded boot loader is located in the System memory, programmed by ST during production. It is used to reprogram the flash memory using one of the following serial interfaces:

- USART on pins PA14/PA15 or PA9/PA10
- I2C on pins PB6/PB7 (STM32F070xx and STM32F030xC devices only)
- USB DFU interface (STM32F070xx devices only)

For further details, refer to the application note AN2606.
3 Embedded flash memory

3.1 Flash main features

- Up to 256 Kbyte of flash memory
- Memory organization:
  - Main flash memory block:
    Up to 64 Kword (64 K × 32 bits)
  - Information block:
    Up to 3 Kword (3 K × 32 bits) for the system memory
    Up to 2 x 8 byte for the option byte

Flash memory interface features:
- Read interface with prefetch buffer
- Option byte loader
- Flash program / erase operation
- Read / write protection
- Low-power mode

3.2 Flash memory functional description

3.2.1 Flash memory organization

The flash memory is organized as 32-bit wide memory cells that can be used for storing both code and data constants.

The memory organization of STM32F030x4, STM32F030x6, STM32F070x6 and STM32F030x8 devices is based on a main flash memory block containing up to 64 pages of 1 Kbyte or up to 16 sectors of 4 Kbytes (4 pages). The sector is the granularity of the write protection (see Section 3.3).

The memory organization of STM32F070xB and STM32F030xC devices is based on a main flash memory block containing up to 128 pages of 2 Kbytes or up to 64 sectors of 4 Kbytes (2 pages). The sector is the granularity of the write protection (see Section 3.3).

The information block is divided into two parts:
1. System memory: used to boot the device in System memory boot mode. The area is reserved for use by STMicroelectronics and contains the boot loader, which is used to reprogram the flash memory through the selected communication interface. It is programmed by ST when the device is manufactured, and protected against spurious write/erase operations. For further details refer to AN2606.
2. Option byte
### Table 4. Flash memory organization (STM32F030x4, STM32F030x6, STM32F070x6 and STM32F030x8 devices)

<table>
<thead>
<tr>
<th>Flash area</th>
<th>Flash memory addresses</th>
<th>Size (byte)</th>
<th>Name</th>
<th>Description(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main flash memory</td>
<td>0x0800 0000 - 0x0800 03FF</td>
<td>1 Kbyte</td>
<td>Page 0</td>
<td>Sector 0</td>
</tr>
<tr>
<td></td>
<td>0x0800 0400 - 0x0800 07FF</td>
<td>1 Kbyte</td>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 0800 - 0x0800 0BFF</td>
<td>1 Kbyte</td>
<td>Page 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 0C00 - 0x0800 0FFF</td>
<td>1 Kbyte</td>
<td>Page 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 7000 - 0x0800 73FF</td>
<td>1 Kbyte</td>
<td>Page 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 7400 - 0x0800 77FF</td>
<td>1 Kbyte</td>
<td>Page 29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 7800 - 0x0800 7BFF</td>
<td>1 Kbyte</td>
<td>Page 30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 7C00 - 0x0800 7FFF</td>
<td>1 Kbyte</td>
<td>Page 31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 F000 - 0x0800 F3FF</td>
<td>1 Kbyte</td>
<td>Page 60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 F400 - 0x0800 F7FF</td>
<td>1 Kbyte</td>
<td>Page 61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 F800 - 0x0800 FBFF</td>
<td>1 Kbyte</td>
<td>Page 62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0800 FC00 - 0x0800 FFFF</td>
<td>1 Kbyte</td>
<td>Page 63</td>
<td></td>
</tr>
<tr>
<td>Information block</td>
<td>0x1FFF EC00 - 0x1FFF F7FF</td>
<td>3 Kbyte(2)</td>
<td>-</td>
<td>System memory</td>
</tr>
<tr>
<td></td>
<td>0x1FFF C400 -0x1FFF F7FF</td>
<td>13 Kbyte(3)</td>
<td>-</td>
<td>System memory</td>
</tr>
<tr>
<td></td>
<td>0x1FFF F800 - 0x1FFF F80F</td>
<td>2 x 8 byte</td>
<td>-</td>
<td>Option byte</td>
</tr>
</tbody>
</table>

1. On STM32F030x4 devices, the main Flash memory space is limited to sector 3. On STM32F030x6 and STM32F070x6 devices, the main Flash memory is limited to sector 7.
2. STM32F030x4, STM32F030x6 and STM32F030x8 devices.
3. STM32F070x6 devices.
Read operations

The embedded flash module can be addressed directly, as a common memory space. Any data read operation accesses the content of the flash module through dedicated read senses and provides the requested data.

The instruction fetch and the data access are both done through the same AHB bus. Read accesses can be performed with the following options managed through the flash access control register (FLASH_ACR):

- Instruction fetch: Prefetch buffer enabled for a faster CPU execution
- Latency: number of wait states for a correct read operation (from 0 to 1)

Instruction fetch

The Arm® Cortex®-M0 fetches the instruction over the AHB bus. The prefetch block aims at increasing the efficiency of instruction fetching.

Prefetch buffer

The prefetch buffer is 3-block wide where each block consists of 4 bytes. The prefetch blocks are direct-mapped. A block can be completely replaced on a single read to the flash memory as the size of the block matches the bandwidth of the flash memory.

The implementation of this prefetch buffer makes a faster CPU execution possible as the CPU fetches one word at a time with the next word readily available in the prefetch buffer. This implies that the acceleration ratio is of the order of 2, assuming that the code is aligned at a 32-bit boundary for the jumps.

### Table 5. Flash memory organization (STM32F070xB, STM32F030xC devices)

<table>
<thead>
<tr>
<th>Flash area</th>
<th>Flash memory addresses</th>
<th>Size (byte)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main flash memory</td>
<td>0x0800 0000 - 0x0800 07FF</td>
<td>2 Kbytes</td>
<td>Page 0</td>
<td>Sector 0</td>
</tr>
<tr>
<td></td>
<td>0x0800 0800 - 0x0800 0FFF</td>
<td>2 Kbytes</td>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0801 F000 - 0x0801 F7FF</td>
<td>2 Kbytes</td>
<td>Page 62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0801 F800 - 0x0801 FFFF</td>
<td>2 Kbytes</td>
<td>Page 63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0803 F000 - 0x0803 F7FF</td>
<td>2 Kbytes</td>
<td>Page 126</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0803 F800 - 0x0803 FFFF</td>
<td>2 Kbytes</td>
<td>Page 127</td>
<td></td>
</tr>
<tr>
<td>Information block</td>
<td>0x1FFF C800 - 0x1FFF F7FF</td>
<td>12 Kbytes(2)</td>
<td>-</td>
<td>System memory</td>
</tr>
<tr>
<td></td>
<td>0x1FFF D800 - 0x1FFF F7FF</td>
<td>8 Kbytes(3)</td>
<td>-</td>
<td>System memory</td>
</tr>
<tr>
<td></td>
<td>0x1FFF F800 - 0x1FFF F80F</td>
<td>2 x 8 byte</td>
<td>-</td>
<td>Option byte</td>
</tr>
</tbody>
</table>

1. The main flash memory space of STM32F070xB is limited to sector 31.
2. STM32F070xB devices only.
3. STM32F030xC devices only.
However the prefetch buffer has an impact on the performance only when the wait state number is 1. In the other case (no wait state) the performance remains the same whatever the prefetch buffer status. There could be some impacts on the power consumption but this is strongly dependent from the actual application code.

**Prefetch controller**

The prefetch controller decides to access the flash memory depending on the available space in the prefetch buffer. The Controller initiates a read request when there is at least one block free in the prefetch buffer.

After reset, the state of the prefetch buffer is on.

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.

**Access latency**

In order to maintain the control signals to read the flash memory, the ratio of the prefetch controller clock period to the access time of the flash memory has to be programmed in the flash access control register with the LATENCY[2:0] bits. This value gives the number of cycles needed to maintain the control signals of the flash memory and correctly read the required data. After reset, the value is zero and only one cycle without additional wait states is required to access the flash memory.

### 3.2.2 Flash program and erase operations

The STM32F0x0 embedded flash memory can be programmed using in-circuit programming or in-application programming.

The **in-circuit programming (ICP)** method is used to update the entire contents of the flash memory, using the SWD protocol or the boot loader to load the user application into the microcontroller. ICP offers quick and efficient design iterations and eliminates unnecessary package handling or socketing of devices.

In contrast to the ICP method, **in-application programming (IAP)** can use any communication interface supported by the microcontroller (I/Os, USB, USART, I²C, SPI, etc.) to download programming data into memory. IAP allows the user to re-program the flash memory while the application is running. Nevertheless, part of the application has to have been previously programmed in the flash memory using ICP.

The program and erase operations can be performed over the whole product voltage range. They are managed through the following seven flash registers:

- Key register (FLASH_KEYR)
- Option byte key register (FLASH_OPTKEYR)
- Flash control register (FLASH_CR)
- Flash status register (FLASH_SR)
- Flash address register (FLASH_AR)
- Option byte register (FLASH_OBR)
- Write protection register (FLASH_WRPR)

An ongoing flash memory operation does not block the CPU as long as the CPU does not access the flash memory.
On the contrary, during a program/erase operation to the flash memory, any attempt to read
the flash memory stalls the bus. The read operation proceeds correctly once the
program/erase operation has completed. This means that code or data fetches cannot be
made while a program/erase operation is ongoing.

For program and erase operations on the flash memory (write/erase), the internal RC
oscillator (HSI) must be ON.

Unlocking the flash memory

After reset, the flash memory is protected against unwanted write or erase operations. The
FLASH_CR register is not accessible in write mode, except for the OBL_LAUNCH bit, used
to reload the option bits. An unlocking sequence should be written to the FLASH_KEYR
register to open the access to the FLASH_CR register. This sequence consists of two write
operations:

- Write KEY1 = 0x45670123
- Write KEY2 = 0xCDEF89AB

Any wrong sequence locks up the FLASH_CR register until the next reset.

In the case of a wrong key sequence, a bus error is detected and a Hard Fault interrupt is
generated. This is done after the first write cycle if KEY1 does not match, or during the
second write cycle if KEY1 has been correctly written but KEY2 does not match.

The FLASH_CR register can be locked again by user software by writing the LOCK bit in the
FLASH_CR register to 1.

For code example refer to the Appendix section A.2.1: Flash memory unlocking sequence.

Main flash memory programming

The main flash memory can be programmed 16 bits at a time. The program operation is
started when the CPU writes a half-word into a main flash memory address with the PG bit
of the FLASH_CR register set. Any attempt to write data that are not half-word long results
in a bus error generating a Hard Fault interrupt.
The flash memory interface preliminarily reads the value at the addressed main flash memory location and checks that it has been erased. If not, the program operation is skipped and a warning is issued by the PGERR bit in FLASH_SR register. The only exception to this is when 0x0000 is programmed. In this case, the location is correctly programmed to 0x0000 and the PGERR bit is not set.

If the addressed main flash memory location is write-protected by the FLASH_WRPR register, the program operation is skipped and a warning is issued by the WRPRTERR bit in the FLASH_SR register. The end of the program operation is indicated by the EOP bit in the FLASH_SR register.
The main flash memory programming sequence in standard mode is as follows:

1. Check that no main flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register.
2. Set the PG bit in the FLASH_CR register.
3. Perform the data write (half-word) at the desired address.
4. Wait until the BSY bit is reset in the FLASH_SR register.
5. Check the EOP flag in the FLASH_SR register (it is set when the programming operation has succeeded), and then clear it by software.

Note: The registers are not accessible in write mode when the BSY bit of the FLASH_SR register is set.

For code example refer to the Appendix section A.2.2: Main flash memory programming sequence.

Flash memory erase

The flash memory can be erased page by page or completely (mass erase).

Page erase

To erase a page, the procedure below should be followed:

1. Check that no flash memory operation is ongoing, by checking the BSY bit in the FLASH_CR register.
2. Set the PER bit in the FLASH_CR register.
3. Program the FLASH_AR register to select a page to erase.
4. Set the STRT bit in the FLASH_CR register (see note below).
5. Wait for the BSY bit to be reset.
6. Check the EOP flag in the FLASH_SR register (it is set when the erase operation has succeeded).
7. Clear the EOP flag.

Note: The software should start checking if the BSY bit equals “0” at least one CPU cycle after setting the STRT bit.

For code example refer to the Appendix section A.2.3: Page erase sequence.
Figure 4. Flash memory Page erase procedure

1. Read LOCK bit in FLASH_CR
2. Check LOCK bit in FLASH_CR = 1
   - Yes: Perform unlock sequence
   - No: Write PER bit in FLASH_CR
3. Write into FAR an address within the page to erase
4. Write STRT bit in FLASH_CR to 1
5. Check BSY bit in FLASH_SR = 1
   - Yes
   - No: Check the page is erased by reading all the addresses in the page
Mass erase

The mass erase command can be used to completely erase the pages of the main flash memory. The information block is unaffected by this procedure. The following sequence is recommended:

1. Check that no flash memory operation is ongoing by checking the BSY bit in the FLASH SR register.
2. Set the MER bit in the FLASH CR register.
3. Set the STRT bit in the FLASH CR register.
4. Wait until the BSY bit is reset in the FLASH SR register.
5. Check the EOP flag in the FLASH SR register (it is set when the programming operation has succeeded).
6. Clear the EOP flag.

_Howard_ 

The software should start checking if the BSY bit equals “0” at least one CPU cycle after setting the STRT bit.

For code example refer to the Appendix section A.2.4: Mass erase sequence.

**Figure 5. Flash memory mass erase procedure**
Option byte programming

The option byte are programmed differently from normal user addresses. The number of option byte is limited to 8 (1, 2 or 4 for write protection, 1 for read protection, 1 for hardware configuration and 2 free byte for user data). After unlocking the flash access, the user has to authorize the programming of the option byte by writing the same set of KEYS (KEY1 and KEY2) to the FLASH_OPTKEYR register to set the OPTWRE bit in the FLASH_CR register (refer to Unlocking the flash memory for key values). Then the user has to set the OPTPG bit in the FLASH_CR register and perform a half-word write operation at the desired flash address.

The value of the addressed option byte is first read to check it is really erased. If not, the program operation is skipped and a warning is issued by the PGERR bit in the FLASH_SR register. The end of the program operation is indicated by the EOP bit in the FLASH_SR register.

The option byte is automatically complemented into the next flash memory address before the programming operation starts. This guarantees that the option byte and its complement are always correct.

The sequence is as follows:
1. Check that no flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register.
2. Unlock the OPTWRE bit in the FLASH_CR register.
3. Set the OPTPG bit in the FLASH_CR register.
4. Write the data (half-word) to the desired address.
5. Wait for the BSY bit to be reset.
6. Read the programmed value and verify.

For code example refer to the Appendix section A.2.6: Option byte programming sequence.

When the flash memory read protection option is changed from protected to unprotected, a mass erase of the main flash memory is performed before reprogramming the read protection option. If the user wants to change an option other than the read protection option, then the mass erase is not performed. The erased state of the read protection option byte protects the flash memory.

Erase procedure

The option byte erase sequence is as follows:
1. Check that no flash memory operation is ongoing by reading the BSY bit in the FLASH_SR register
2. Unlock the OPTWRE bit in the FLASH_CR register
3. Set the OPTER bit in the FLASH_CR register
4. Set the STRT bit in the FLASH_CR register
5. Wait for the BSY bit to be reset
6. Read the erased option byte and verify

For code example refer to the Appendix section A.2.7: Option byte erasing sequence.
3.3 Memory protection

The user area of the flash memory can be protected against read by untrusted code. The pages of the flash memory can also be protected against unwanted write due to loss of program counter contexts. The write-protection granularity is one sector (four pages).

3.3.1 Read protection

The read protection is activated by setting the RDP option byte then by applying the system reset to reload the new RDP option byte.

Note: Instead of the system reset, apply the power-on reset (POR) to reload the new RDP option byte if the debugger connection occurred since the last POR. If the read protection is programmed through software, do not set the OBL_LAUNCH bit (FLASH_CR register) but perform a POR to reload the option bytes. This can be done by Standby or Shutdown mode entry followed by a wakeup.

There are three levels of read protection from no protection (level 0) to maximum protection or no debug (level 2). Refer to Table 7: Access status versus protection level and execution modes.

The flash memory is protected when the RDP option byte and its complement contain the pair of values shown in Table 6.

Table 6. Flash memory read protection status

<table>
<thead>
<tr>
<th>RDP byte value</th>
<th>RDP complement value</th>
<th>Read protection level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xAA</td>
<td>0x55</td>
<td>Level 0 (ST production configuration)</td>
</tr>
<tr>
<td>Any value except 0xAA or 0xCC</td>
<td>Any value (not necessarily complementary) except 0x55 and 0x33</td>
<td>Level 1</td>
</tr>
<tr>
<td>0xCC</td>
<td>0x33</td>
<td>Level 2</td>
</tr>
</tbody>
</table>

The System memory area is read accessible whatever the protection level. It is never accessible for program/erase operation.

Level 0: no protection

Read, program and erase operations into the main flash memory area are possible.

The option byte are as well accessible by all operations.

Level 1: read protection

This is the default protection level when RDP option byte is erased. It is defined as well when RDP value is at any value different from 0xAA and 0xCC, or even if the complement is not correct.

- **User mode**: Code executing in user mode can access main flash memory and option byte with all operations.
- **Debug, boot RAM and boot loader modes**: In debug mode (with SWD) or when code is running from boot RAM or boot loader, the main flash memory is totally inaccessible. In these modes, even a simple read access generates a bus error and a Hard Fault interrupt. The main flash memory is program/erase protected to prevent malicious or
unauthorized users from reprogramming any of the user code with a dump routine. Any attempted program/erase operation sets the PGERR flag of flash status register (FLASH_SR).

When the RPD is reprogrammed to the value 0xAA to move back to Level 0, a mass erase of the main flash memory is performed.

**Level 2: no debug**

In this level, the protection level 1 is guaranteed. In addition, the CortexM0 debug capabilities are disabled. Consequently, the debug port (SWD), the boot from RAM (boot RAM mode) and the boot from System memory (boot loader mode) are no more available.

In user execution mode, all operations are allowed on the main flash memory. On the contrary, only read and program operations can be performed on the option byte. Option byte are not accessible for erase operations.

Moreover, the RDP byte cannot be programmed. Thus, the level 2 cannot be removed at all: it is an irreversible operation. When attempting to program the RDP byte, the protection error flag WRPRTERR is set in the flash_SR register and an interrupt can be generated.

*Note: The debug feature is also disabled under reset.*

*STM* is not able to perform analysis on defective parts on which the level 2 protection has been set.

**Table 7. Access status versus protection level and execution modes**

<table>
<thead>
<tr>
<th>Area</th>
<th>Protection level</th>
<th>User execution</th>
<th>Debug / Boot from RAM / Boot from System memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>Main flash memory</td>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>System memory(2)</td>
<td>1</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Option byte</td>
<td>1</td>
<td>Yes</td>
<td>Yes(^{(3)})</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Yes</td>
<td>Yes(^{(5)})</td>
</tr>
</tbody>
</table>

1. When the protection level 2 is active, the Debug port, the boot from RAM and the boot from System memory are disabled.
2. The system memory is only read-accessible, whatever the protection level (0, 1 or 2) and execution mode.
3. The main flash memory is erased when the RDP option byte is changed from level 1 to level 0 (0xAA).
4. When the RDP level 1 is active, the embedded boot loader does not allow to read or write the Option byte, except to remove the RDP protection (move from level 1 to level 0).
5. All option byte can be programmed, except the RDP byte.

**Changing read protection level**

It is easy to move from level 0 to level 1 by changing the value of the RDP byte to any value (except 0xCC).

By programming the 0xCC value in the RDP byte, it is possible to go to level 2 either directly from level 0 or from level 1.
On the contrary, the change to level 0 (no protection) is not possible without a main flash memory Mass erase operation. This Mass erase is generated as soon as 0xAA is programmed in the RDP byte.

Note: To validate the protection level change, the option byte must be reloaded through the "OBL_LAUNCH" bit in Flash control register.

### 3.3.2 Write protection

The write protection is implemented with a granularity of one sector. It is activated by configuring the WRPx option byte, and then by reloading them by setting the OBL_LAUNCH bit in the FLASH_CR register.

If a program or an erase operation is performed on a protected sector, the flash memory returns a WRPRTERR protection error flag in the flash memory Status Register (FLASH_SR).

#### Write unprotection

To disable the write protection, two application cases are provided:

- **Case 1:** Read protection disabled after the write unprotection:
  - Erase the entire option byte area by using the OPTER bit in the flash memory control register (FLASH_CR).
  - Program the code 0xAA in the RDP byte to unprotect the memory. This operation forces a mass erase of the main flash memory.
  - Set the OBL_LAUNCH bit in the flash control register (FLASH_CR) to reload the option byte (and the new WRP[1:0] byte), and to disable the write protection.

- **Case 2:** Read protection maintained active after the write unprotection, useful for in-application programming with a user boot loader:
  - Erase the entire option byte area by using the OPTER bit in the flash memory control register (FLASH_CR).
  - Set the OBL_LAUNCH bit in the flash control register (FLASH_CR) to reload the option byte (and the new WRP[1:0] byte), and to disable the write protection.

### 3.3.3 Option byte write protection

The option byte are always read-accessible and write-protected by default. To gain write access (Program/Erase) to the option byte, a sequence of keys (same as for lock) has to be written into the OPTKEYR. A correct sequence of keys gives write access to the option byte and this is indicated by OPTWRE in the FLASH_CR register being set. Write access can be disabled by resetting the bit through software.

### 3.4 Flash interrupts

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of operation</td>
<td>EOP</td>
<td>EOIPE</td>
</tr>
<tr>
<td>Write protection error</td>
<td>WRPRTERR</td>
<td>ERRIE</td>
</tr>
<tr>
<td>Programming error</td>
<td>PGERR</td>
<td>ERRIE</td>
</tr>
</tbody>
</table>
3.5 Flash register description

The flash memory registers have to be accessed by 32-bit words (half-word and byte accesses are not allowed).

3.5.1 Flash access control register (FLASH_ACR)

Address offset: 0x00
Reset value: 0x0000 0000

The table below shows the bit fields of the Flash access control register (FLASH_ACR):

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
<td>Must be kept at reset value.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>PRFTBS</td>
<td>Prefetch buffer status. This bit provides the status of the prefetch buffer.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>PRFTBE</td>
<td>Prefetch buffer enable. 0: Prefetch is disabled. 1: Prefetch is enabled.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>LATENCY[2:0]</td>
<td>Latency. These bits represent the ratio of the SYSCLK (system clock) period to the flash access time.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Bits 31:6 Reserved, must be kept at reset value.

Bit 5 **PRFTBS**: Prefetch buffer status
- 0: Prefetch buffer is disabled
- 1: Prefetch buffer is enabled

*Note: The prefetch status is set to 1 as soon a first fetch request is done*

Bit 4 **PRFTBE**: Prefetch buffer enable
- 0: Prefetch is disabled
- 1: Prefetch is enabled

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **LATENCY[2:0]**: Latency
- 000: Zero wait state, if SYSCLK ≤ 24 MHz
- 001: One wait state, if 24 MHz < SYSCLK ≤ 48 MHz
- 010: Two wait states, if 48 MHz < SYSCLK
- 111: Three wait states, if 72 MHz < SYSCLK

3.5.2 Flash key register (FLASH_KEYR)

Address offset: 0x04
Reset value: 0xXXXX XXXX

All these register bits are write-only and return a 0 when read.

The table below shows the bit fields of the Flash key register (FLASH_KEYR):

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>FKEY[31:16]</td>
<td>Flash key</td>
</tr>
<tr>
<td>15</td>
<td>FKEY[15:0]</td>
<td>Flash key</td>
</tr>
</tbody>
</table>

Bits 31:0 **FKEY**: Flash key
- These bits represent the keys to unlock the flash.
3.5.3 Flash option key register (FLASH_OPTKEYR)

Address offset: 0x08
Reset value: 0xXXXX XXXX

All these register bits are write-only and return a 0 when read.

<table>
<thead>
<tr>
<th>OPTKEY[31:16]</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OPTKEY[15:0]</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>

Bits 31:0 OPTKEY: Option byte key
These bits represent the keys to unlock the OPTWRE.
### 3.5.4 Flash status register (FLASH_SR)

Address offset: 0x0C
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>rc_w1</td>
<td>rc_w1</td>
<td>rc_w1</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:6 Reserved, must be kept at reset value.

- **Bit 5 EOP**: End of operation
  - Set by hardware when a flash operation (programming / erase) is completed.
  - Reset by writing 1.
  - *Note: EOP is asserted at the end of each successful program or erase operation*

- **Bit 4 WRPRTERR**: Write protection error
  - Set by hardware when programming a write-protected address of the flash memory.
  - Reset by writing 1.

- **Bit 3** Reserved, must be kept at reset value.

- **Bit 2 PGERR**: Programming error
  - Set by hardware when an address to be programmed contains a value different from '0xFFFF' before programming.
  - Reset by writing 1.
  - *Note: The STRT bit in the FLASH_CR register should be reset before starting a programming operation.*

- **Bit 1** Reserved, must be kept at reset value.

- **Bit 0 BSY**: Busy
  - This indicates that a flash operation is in progress. This is set on the beginning of a flash operation and reset when the operation finishes or when an error occurs.

### 3.5.5 Flash control register (FLASH_CR)

Address offset: 0x10
Reset value: 0x0000 0080

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>OBL_LAUNCH</td>
<td>EOPIE</td>
<td>ERRIE</td>
<td>OPTWRE</td>
<td>LOCK</td>
<td>STRT</td>
<td>OPTER</td>
<td>OPTPG</td>
<td>MER</td>
<td>PER</td>
<td>PG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:14 Reserved, must be kept at reset value.
Bit 13 **OBL_LAUNCH**: Force option byte loading
When set to 1, this bit forces the option byte reloading. This operation generates a system reset.
- 0: Inactive
- 1: Active

Bit 12 **EOPIE**: End of operation interrupt enable
This bit enables the interrupt generation when the EOP bit in the FLASH_SR register goes to 1.
- 0: Interrupt generation disabled
- 1: Interrupt generation enabled

Bit 11 Reserved, must be kept at reset value

Bit 10 **ERRIE**: Error interrupt enable
This bit enables the interrupt generation on an error when PGERR / WRPRTERR are set in the FLASH_SR register.
- 0: Interrupt generation disabled
- 1: Interrupt generation enabled

Bit 9 **OPTWRE**: Option byte write enable
When set, the option byte can be programmed. This bit is set on writing the correct key sequence to the FLASH_OPTKEYR register.
This bit can be reset by software

Bit 8 Reserved, must be kept at reset value.

Bit 7 **LOCK**: Lock
Write to 1 only. When it is set, it indicates that the flash is locked. This bit is reset by hardware after detecting the unlock sequence.
In the event of unsuccessful unlock operation, this bit remains set until the next reset.

Bit 6 **STRT**: Start
This bit triggers an ERASE operation when set. This bit is set only by software and reset when the BSY bit is reset.

Bit 5 **OPTER**: Option byte erase
Option byte erase chosen.

Bit 4 **OPTPG**: Option byte programming
Option byte programming chosen.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **MER**: Mass erase
Erase of all user pages chosen.

Bit 1 **PER**: Page erase
Page erase chosen.

Bit 0 **PG**: Programming
Flash programming chosen.
### 3.5.6 Flash address register (FLASH_AR)

Address offset: 0x14  
Reset value: 0x0000 0000

This register is updated by hardware with the currently/last used address. For Page erase operations, this should be updated by software to indicate the chosen page.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 31:0 **FAR**: Flash address  
Chooses the address to program when programming is selected, or a page to erase when Page erase is selected.  
**Note:** Write access to this register is blocked when the BSY bit in the FLASH_SR register is set.

### 3.5.7 Flash Option byte register (FLASH_OBR)

Address offset 0x1C  
Reset value: 0xXXXX XX0X

The reset value of this register depends on the value programmed in the option byte.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

Bits 31:24 **DATA1**  
Bits 23:16 **DATA0**  
Bit 15 Reserved, must be kept at reset value  
Bits 14:12 User option byte:  
Bit 14: **RAM_PARITY_CHECK**  
Bit 13: **VDDA_MONITOR**  
Bit 12: **nBOOT1**  
Bit 11 Reserved, must be kept at reset value  
Bits 10:8 Bit 10: **nRST_STDBY**  
Bit 9: **nRST_STOP**  
Bit 8: **WDG_SW**
Bits 7:3 Reserved, must be kept at reset value.

Bits 2:1 **RDPRT[1:0]**: Read protection level status
- 00: Read protection level 0 is enabled (ST production configuration)
- 01: Read protection level 1 is enabled
- 11: Read protection level 2 is enabled.

Bit 0 **OPTERR**: Option byte error
When set, this indicates that the loaded option byte and its complement do not match. The corresponding byte is set to 0xFF in respective FLASH_OBR or FLASH_WRPR register.

3.5.8 Write protection register (FLASH_WRPR)

Address offset: 0x20
Reset value: 0xXXXX XXXX

The reset value of this register depends on the value programmed in the option byte.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WRP[31:16]**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WRP[15:0]**

Bits 31:0 **WRP**: Write protect
This register contains the write-protection option byte loaded by the OBL.
### 3.5.9 Flash register map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Description</th>
<th>Reset value</th>
<th>Latency [2:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>FLASH_ACR</td>
<td>Off-set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x004</td>
<td>FLASH_KEYR</td>
<td>FKEY[31:0]</td>
<td>0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0x008</td>
<td>FLASH_OPTKEYR</td>
<td>OPTKEY[31:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00C</td>
<td>FLASH_SR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x010</td>
<td>FLASH_CR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x014</td>
<td>FLASH_AR</td>
<td>FAR[31:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01C</td>
<td>FLASH_OBR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x020</td>
<td>FLASH_WRPR</td>
<td>WRP[31:0]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refer to Section 2.2 on page 37 for the register boundary addresses.
4 Option bytes

There are up to 8 option bytes. They are configured by the end user depending on the application requirements. As a configuration example, the watchdog may be selected in hardware or software mode.

A 32-bit word is split up as follows in the option byte.

<table>
<thead>
<tr>
<th>31-24</th>
<th>23-16</th>
<th>15-8</th>
<th>7-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complemented option byte 1</td>
<td>Option byte 1</td>
<td>Complemented option byte 0</td>
<td>Option byte 0</td>
</tr>
</tbody>
</table>

The organization of these bytes inside the information block is as shown in Table 11.

The option byte can be read from the memory locations listed in Table 11 or from the Option byte register (FLASH_OBR).

Note: The new programmed option byte (user, read/write protection) are not loaded after a system reset. To reload them, either a POR or setting to ‘1’ the OBL_LAUNCH bit is necessary.

<table>
<thead>
<tr>
<th>Address</th>
<th>[31:24]</th>
<th>[23:16]</th>
<th>[15:8]</th>
<th>[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1FFF F800</td>
<td>nUSER</td>
<td>USER</td>
<td>nRDP</td>
<td>RDP</td>
</tr>
<tr>
<td>0x1FFF F804</td>
<td>nData1</td>
<td>Data1</td>
<td>nData0</td>
<td>Data0</td>
</tr>
<tr>
<td>0x1FFF F808</td>
<td>nWRP1</td>
<td>WRP1</td>
<td>nWRP0</td>
<td>WRP0</td>
</tr>
<tr>
<td>0x1FFF F80C</td>
<td>nWRP3</td>
<td>WRP3</td>
<td>nWRP2</td>
<td>WRP2</td>
</tr>
</tbody>
</table>

On every power-on reset, the option byte loader (OBL) reads the information block and stores the data into the option byte register (FLASH_OBR) and the write protection register (FLASH_WRPR). During option byte loading, the bit-wise complementarity of the option byte and its corresponding complemented option byte is verified. In case of failure, an option byte error (OPTERR) is generated and the corresponding option byte is considered as 0xFF. If the option byte and its complemented option byte are both equal to 0xFF (Electrical Erase state) the option byte error is not generated.
## 4.1 Option byte description

### 4.1.1 User and read protection option byte

Flash memory address: 0x1FFF F800  
ST production value: 0x00FF 55AA

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>nUSER</td>
<td>Res.</td>
<td>RAM_PARITY_CHECK</td>
<td>VDDA_MONITOR</td>
<td>nBOOT1</td>
<td>Res.</td>
<td>nRST_STDBY</td>
<td>nRST_STOP</td>
<td>WDG_SW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRDP</td>
<td>Res.</td>
<td>RDP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:24 **nUSER**: User option byte complement

- Bit 23 Reserved, must be kept as “1”

Bits 22:20 **USER**: User option byte (stored in FLASH_OBR[15:8])

- Bit 22: **RAM_PARITY_CHECK**
  - 0: RAM parity check enabled
  - 1: RAM parity check disabled

- Bit 21: **VDDA_MONITOR**
  - 0: VDDA power supply supervisor disabled
  - 1: VDDA power supply supervisor enabled

- Bit 20: **nBOOT1**
  - Together with the BOOT0 signal, it selects the device boot mode. Refer to Section 2.5: Boot configuration for more details.

- Bit 19 Reserved, must be kept as “1”
4.1.2 User data option byte

Flash memory address: 0x1FFF F804
ST production value: 0x00FF 00FF

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>nData1</td>
<td>Data1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| nData0 | Data0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:24 nData1: User data byte 1 complement

Bits 23:16 Data1: User data byte 1 value (stored in FLASH_OBR[31:24])

Bits 15:8 nData0: User data byte 0 complement

Bits 7:0 Data0: User data byte 0 value (stored in FLASH_OBR[23:16])

4.1.3 Write protection option byte

This set of registers is used to write-protect the flash memory. Clearing a bit in WRPx field (and at the same time setting a corresponding bit in nWRPx field) write-protects the given memory sector.

For STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices, WRP bits from 0 to 31 are protecting the flash memory by sector of 4 kB.

For STM32F030xC devices, WRP bits from 0 to 30 are protecting the first 124 kB by sector of 4 kB and the bit 31 is protecting the last 132 kB.

Refer to Section 3.3.2: Write protection for more details.
Flash memory address: 0x1FFF F808
ST production value: 0x00FF 00FF

<table>
<thead>
<tr>
<th>Offset</th>
<th>Option byte</th>
<th>Option byte map and ST production values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>User and read protection</td>
<td>nUSER</td>
</tr>
<tr>
<td></td>
<td>nUSER</td>
<td>Res.</td>
</tr>
<tr>
<td></td>
<td>nUSER</td>
<td>nBOOT1</td>
</tr>
<tr>
<td></td>
<td>nBOOT1</td>
<td>Res.</td>
</tr>
<tr>
<td></td>
<td>nRST, STOP</td>
<td>WDG, SW</td>
</tr>
<tr>
<td>ST production value</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
<td>0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>0x04</td>
<td>User data</td>
<td>nData1</td>
</tr>
<tr>
<td></td>
<td>nData1</td>
<td>Data1</td>
</tr>
<tr>
<td></td>
<td>nData0</td>
<td>nData0</td>
</tr>
<tr>
<td>ST production value</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0x08</td>
<td>Write protection</td>
<td>nWRP1</td>
</tr>
<tr>
<td></td>
<td>nWRP1</td>
<td>WRP1</td>
</tr>
<tr>
<td></td>
<td>nWRP0</td>
<td>nWRP0</td>
</tr>
<tr>
<td>ST production value</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0x0C</td>
<td>Write protection</td>
<td>nWRP3</td>
</tr>
<tr>
<td></td>
<td>nWRP3</td>
<td>WRP3</td>
</tr>
<tr>
<td></td>
<td>nWRP2</td>
<td>nWRP2</td>
</tr>
<tr>
<td>ST production value</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
5 Cyclic redundancy check calculation unit (CRC)

5.1 Introduction

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from 8-, 16- or 32-bit data word and a generator polynomial.

Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the functional safety standards, 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.

5.2 CRC main features

• Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7
  \[x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1\]
• Handles 8-, 16-, 32-bit data size
• Programmable CRC initial value
• Single input/output 32-bit data register
• Input buffer to avoid bus stall during calculation
• CRC computation done in 4 AHB clock cycles (HCLK) for the 32-bit data size
• General-purpose 8-bit register (can be used for temporary storage)
• Reversibility option on I/O data
• Accessed through AHB slave peripheral by 32-bit words only, with the exception of CRC_DR register that can be accessed by words, right-aligned half-words and right-aligned bytes
5.3 CRC functional description

5.3.1 CRC block diagram

![CRC calculation unit block diagram](image)

5.3.2 CRC internal signals

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Signal type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crc_hclk</td>
<td>Digital input</td>
<td>AHB clock</td>
</tr>
</tbody>
</table>

5.3.3 CRC operation

The CRC calculation unit has a single 32-bit read/write data register (CRC_DR). It is used to input new data (write access), and holds the result of the previous CRC calculation (read access).

Each write operation to the data register creates a combination of the previous CRC value (stored in CRC_DR) and the new one. CRC computation is done on the whole 32-bit data word or byte by byte depending on the format of the data being written.

The CRC_DR register can be accessed by word, right-aligned half-word and right-aligned byte. For the other registers only 32-bit accesses are allowed.

The duration of the computation depends on data width:
- 4 AHB clock cycles for 32 bits
- 2 AHB clock cycles for 16 bits
- 1 AHB clock cycles for 8 bits

An input buffer allows a second data to be immediately written without waiting for any wait states due to the previous CRC calculation.
The data size can be dynamically adjusted to minimize the number of write accesses for a given number of bytes. For instance, a CRC for 5 bytes can be computed with a word write followed by a byte write.

The input data can be reversed to manage the various endianness schemes. The reversing operation can be performed on 8 bits, 16 bits and 32 bits depending on the REV_IN[1:0] bits in the CRC_CR register.

For example, 0x1A2B3C4D input data are used for CRC calculation as:
- 0x58D43CB2 with bit-reversal done by byte
- 0xD458B23C with bit-reversal done by half-word
- 0xB23CD458 with bit-reversal done on the full word

The output data can also be reversed by setting the REV_OUT bit in the CRC_CR register. The operation is done at bit level. For example, 0x11223344 output data are converted to 0x22CC4488.

The CRC calculator can be initialized to a programmable value using the RESET control bit in the CRC_CR register (the default value is 0xFFFFFFFF).

The initial CRC value can be programmed with the CRC_INIT register. The CRC_DR register is automatically initialized upon CRC_INIT register write access.

The CRC_IDR register can be used to hold a temporary value related to CRC calculation. It is not affected by the RESET bit in the CRC_CR register.

### 5.4 CRC registers

The CRC_DR register can be accessed by words, right-aligned half-words and right-aligned bytes. For the other registers only 32-bit accesses are allowed.

#### 5.4.1 CRC data register (CRC_DR)

Address offset: 0x00
Reset value: 0xFFFF FFFF

<table>
<thead>
<tr>
<th>Bits 31:0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DR[31:16]</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DR[15:0]</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:0 **DR[31:0]**: Data register bits
- This register is used to write new data to the CRC calculator.
- It holds the previous CRC calculation result when it is read.
- If the data size is less than 32 bits, the least significant bits are used to write/read the correct value.
### 5.4.2 CRC independent data register (CRC_IDR)

Address offset: 0x04  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 31:8  Reserved, must be kept at reset value.

Bits 7:0  **IDR[7:0]:** General-purpose 8-bit data register bits  
These 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

### 5.4.3 CRC control register (CRC_CR)

Address offset: 0x08  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 31:8  Reserved, must be kept at reset value.

Bit 7  **REV_OUT:** Reverse output data  
This bit controls the reversal of the bit order of the output data.  
0: Bit order not affected  
1: Bit-reversed output format
5.4.4 CRC initial value (CRC_INIT)

Address offset: 0x10

Reset value: 0xFFFF

**CRC_INIT[31:16]**

<p>| | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

**CRC_INIT[15:0]**

<p>| | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bits 6:5 **REV_IN[1:0]**: Reverse input data
This bitfield controls the reversal of the bit order of the input data
00: Bit order not affected
01: Bit reversal done by byte
10: Bit reversal done by half-word
11: Bit reversal done by word

Bits 4:1 Reserved, must be kept at reset value.

Bit 0 **RESET**: RESET bit
This bit is set by software to reset the CRC calculation unit and set the data register to the value stored in the CRC_INIT register. This bit can only be set, it is automatically cleared by hardware.

**CRC_INIT[31:0]**: Programmable initial CRC value
This register is used to write the CRC initial value.
### 5.4.5 CRC register map

Table 14. CRC register map and reset values

| Offset | Register name | 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   | CRC_DR        |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |               | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
|        | Reset value   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x04   | CRC_IDR       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |               | 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   |
|        | Reset value   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x08   | CRC_CR        |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |               | 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   |
|        | Reset value   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x10   | CRC_INIT      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |               | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
|        | Reset value   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

Refer to Section 2.2 on page 37 for the register boundary addresses.
6 Power control (PWR)

6.1 Power supplies

The STM32F030/STM32F070 subfamily embeds a voltage regulator in order to supply the internal 1.8 V digital power domain.

- The STM32F030/STM32F070 devices require a 2.4 V - 3.6 V operating supply voltage ($V_{DD}$) and a 2.4 V - 3.6 V analog supply voltage ($V_{DDA}$).

![Figure 7. Power supply overview](image)

6.1.1 Independent A/D converter supply and reference voltage

To improve conversion accuracy and to extend the supply flexibility, the ADC has an independent power supply that 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 $V_{SSA}$.

The $V_{DDA}$ supply/reference voltage must be equal or higher than $V_{DD}$.

When a single supply is used, $V_{DDA}$ can be externally connected to $V_{DD}$, through the external filtering circuit in order to ensure a noise free $V_{DDA}$ reference voltage.

When $V_{DDA}$ is different from $V_{DD}$, $V_{DDA}$ must always be higher or equal to $V_{DD}$. To keep safe potential difference in between $V_{DDA}$ and $V_{DD}$ during power-up/power-down, an external Shottky diode may be used between $V_{DD}$ and $V_{DDA}$. Refer to the datasheet for the maximum allowed difference.
6.1.2 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.

6.2 Power supply supervisor

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

STM32F0x1xx and STM32F0x2xx devices feature integrated power-on reset (POR) and power-down reset (PDR) circuits, which are always active and ensure proper operation above a threshold of 2 V.

The device remains in Reset mode when the monitored supply voltage is below a specified threshold, $V_{POR/PDR}$, without the need for an external reset circuit.

- The POR monitors only the $V_{DD}$ supply voltage. During the startup phase $V_{DDA}$ must arrive first and be greater than or equal to $V_{DD}$.
- The PDR monitors both the $V_{DD}$ and $V_{DDA}$ supply voltages. However, the $V_{DDA}$ power supply supervisor can be disabled (by programming a dedicated option bit $V_{DDA\_MONITOR}$) to reduce the power consumption if the application is designed to make sure that $V_{DDA}$ is higher than or equal to $V_{DD}$.

For more details on the power on / power down reset threshold, refer to the electrical characteristics section in the datasheet.

Figure 8. Power on reset/power down reset waveform
6.3 Low-power modes

By default, the microcontroller is in Run mode after a system or a power Reset. 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 wake-up sources.

The device features three low-power modes:

- **Sleep mode (CPU clock off, all peripherals including Arm® Cortex®-M0 core peripherals like NVIC, SysTick, etc. are 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 reduced by one of the following means:

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

In Run mode the speed of the system clocks (SYSCLK, HCLK, PCLK) 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 7.4.2: Clock configuration register (RCC_CFGR).
6.3.2 Peripheral clock gating

In Run mode, the AHB clock (HCLK) and the APB clock (PCLK) 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), the APB peripheral clock enable register 2 (RCC_APB2ENR) and the APB peripheral clock enable register 1 (RCC_APB1ENR).

6.3.3 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 Arm® Cortex®-M0 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.

In the Sleep mode, all I/O pins keep the same state as in the Run mode.

Refer to Table 16 and Table 17 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 wake-up 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 Arm® Cortex®-M0 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) must 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 wake-up time as no time is wasted in interrupt entry/exit.

Refer to Table 16 and Table 17 for more details on how to exit Sleep mode.
6.3.4 Stop mode

The Stop mode is based on the Arm® Cortex®-M0 deep sleep 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 oscillators are disabled. SRAM and register contents are preserved.

In the Stop mode, all I/O pins keep the same state as in the Run mode.

Entering Stop mode

Refer to Table 18 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 19.3: IWDG functional description.

<table>
<thead>
<tr>
<th>Mode entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If WFI was used for entry:</td>
<td>Interrupt: Refer to Table 31: Vector table</td>
</tr>
<tr>
<td>If WFE was used for entry</td>
<td>Wake-up event: Refer to Section 11.2.3: Event management</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sleep-on-exit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFI (wait for interrupt) while:</td>
<td>Interrupt: Refer to Table 31: Vector table</td>
</tr>
<tr>
<td>– SLEEPDEEP = 0 and</td>
<td></td>
</tr>
<tr>
<td>– SLEEPONEXIT = 1</td>
<td></td>
</tr>
</tbody>
</table>

### Table 16. Sleep-now

<table>
<thead>
<tr>
<th>Sleep-now mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode entry</td>
<td>WFI (Wait for Interrupt) or WFE (Wait for Event) while:</td>
</tr>
<tr>
<td>– SLEEPDEEP = 0 and</td>
<td></td>
</tr>
<tr>
<td>– SLEEPONEXIT = 0</td>
<td></td>
</tr>
</tbody>
</table>

Refer to the Arm® Cortex®-M0 System Control register.

<table>
<thead>
<tr>
<th>Mode exit</th>
<th>None</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Wake-up latency</th>
<th>None</th>
</tr>
</thead>
</table>

### Table 17. Sleep-on-exit

<table>
<thead>
<tr>
<th>Sleep-on-exit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode entry</td>
<td>WFI (wait for interrupt) while:</td>
</tr>
<tr>
<td>– SLEEPDEEP = 0 and</td>
<td></td>
</tr>
<tr>
<td>– SLEEPONEXIT = 1</td>
<td></td>
</tr>
</tbody>
</table>

Refer to the Arm® Cortex®-M0 System Control register.

<table>
<thead>
<tr>
<th>Mode exit</th>
<th>None</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Wake-up latency</th>
<th>None</th>
</tr>
</thead>
</table>
- real-time clock (RTC): this is configured by the RTCEN bit in the **RTC domain control register (RCC_BDCR)**.
- Internal RC oscillator (LSI): this is configured by the LSION bit in the **Control/status register (RCC_CSR)**.
- External 32.768 kHz oscillator (LSE): this is configured by the LSEON bit in the **RTC domain control register (RCC_BDCR)**.

The ADC can also consume power during Stop mode, unless it is disabled before entering this mode. Refer to **ADC control register (ADC_CR)** for details on how to disable it.

**Exiting Stop mode**

Refer to **Table 18** for more details on how to exit Stop mode.

When exiting Stop mode by issuing an interrupt or a wake-up event, the HSI 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 18. Stop mode

<table>
<thead>
<tr>
<th>Stop mode</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Mode entry** | WFI (Wait for Interrupt) or WFE (Wait for Event) while:  
- Set SLEEPDEEP bit in Arm® Cortex®-M0 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)**), all peripherals interrupt pending bits and RTC Alarm flag must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues.  
  
  If the application needs to disable the external oscillator (external clock) before entering Stop mode, the system clock source must be first switched to HSI and then clear the HSEON bit.  
  
  Otherwise, if before entering Stop mode the HSEON bit is kept at 1, the security system (CSS) feature must be enabled to detect any external oscillator (external clock) failure and avoid a malfunction when entering Stop mode. |
| **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 31: Vector table**.  
  
  If WFE was used for entry:  
  Any EXTI line configured in event mode. Refer to **Section 11.2.3: Event management on page 174** |
| **Wake-up latency** | HSI wake-up time + regulator wake-up time from Low-power mode |
6.3.5 Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Arm® Cortex®-M0 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 Standby circuitry (see Figure 7).

Entering Standby mode

Refer to Table 19 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 19.3: IWDG functional description.
- Real-time clock (RTC): this is configured by the RTCEN bit in the RTC domain control register (RCC_BDCR).
- Internal RC oscillator (LSI): this is configured by the LSION bit in the Control/status register (RCC_CSR).
- External 32.768 kHz oscillator (LSE): this is configured by the LSEON bit in the RTC domain control register (RCC_BDCR).

Exiting Standby mode

The microcontroller exits the Standby mode when an external reset (NRST pin), an IWDG reset, a rising edge on one of the enabled WKUPx pins or an RTC event occurs. All registers are reset after wake-up 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 pin sampling, option bytes loading, reset vector 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 19 for more details on how to exit Standby mode.

| Table 19. Standby mode |
|------------------------|------------------|
| **Mode entry**         | Description                                               |
| WFI (Wait for Interrupt) or WFE (Wait for Event) while: | |
| – Set SLEEPDEEP in Arm® Cortex®-M0 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 event’s rising edge, external Reset in NRST pin, IWDG Reset. |
| **Wake-up latency**    | Reset phase                                                |

Refer to Table 19 for more details on how to enter Standby mode.
I/O states in Standby mode
In Standby mode, all I/O pins are high impedance except:
- Reset pad (still available)
- PC13, PC14 and PC15 if configured by RTC or LSE
- WKUPx pins

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 Arm® Cortex®-M0 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.

6.3.6 RTC wakeup from low-power mode
The RTC can be used to wake-up the MCU from low-power mode by means of the RTC alarm. For this purpose, two of the three alternative RTC clock sources can be selected by programming the RTCSEL[1:0] bits in the RTC 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 1 µA added consumption in typical conditions)
- Low-power internal RC oscillator (LSI)
  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 wake-up 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 wake-up from Standby mode, there is no need to configure the EXTI Line 17.
6.4 Power control registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

6.4.1 Power control register (PWR_CR)

Address offset: 0x00

Reset value: 0x0000 0000 (reset by wake-up from Standby mode)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:9 Reserved, must be kept at reset value.

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

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

0: Access to RTC disabled
1: Access to RTC enabled

Bits 7:4 Reserved, must be kept at reset value

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 wake-up flag.

This bit is always read as 0.

0: No effect
1: Clear the WUF wake-up 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

Note: When a peripheral that can work in STOP mode requires a clock, the Power controller automatically switch the voltage regulator from Low-power mode to Normal mode and remains in this mode until the request disappears.
6.4.2 Power control/status register (PWR_CSR)

Address offset: 0x04

Reset value: 0x0000 000X (not reset by wake-up from Standby mode)

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

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Res. EWUP 7 EWUP 6 EWUP 5 EWUP 4 Res. EWUP 3 EWUP 2 EWUP 1 Res. Res. Res. Res. SBF WUF

Bits 31:15 Reserved, must be kept at reset value.

Bits 14:11 **EWUPx**: Enable WKUPx pin (available only on STM32F070xB and STM32F030xC devices)
These bits are set and cleared by software.
0: WKUPx pin is used for general purpose I/O. An event on the WKUPx pin does not wake-up the device from Standby mode.
1: WKUPx pin is used for wake-up from Standby mode and forced in input pull down configuration (rising edge on WKUPx pin wakes-up the system from Standby mode).

*Note: These bits are reset by a system Reset.*

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **EWUPx**: Enable WKUPx pin
These bits are set and cleared by software.
0: WKUPx pin is used for general purpose I/O. An event on the WKUPx pin does not wake-up the device from Standby mode.
1: WKUPx pin is used for wakeup from Standby mode and forced in input pull down configuration (rising edge on WKUPx pin wakes-up the system from Standby mode).

*Note: These bits are reset by a system Reset.*

Bits 7:2 Reserved, must be kept at reset value.

Bit 1 **SBF**: Standby flag
This bit is set by hardware when the device enters Standby mode and it is 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**: Wake-up flag
This bit is set by hardware to indicate that the device received a wake-up event. It is cleared by a system reset or by setting the CWUF bit in the Power control register (PWR_CR)
0: No wake-up event occurred
1: A wake-up event was received from one of the enabled WKUPx pins or from the RTC alarm.

*Note: An additional wake-up event is detected if one WKUPx pin is enabled (by setting the EWUPx bit) when its pin level is already high.*
### 6.4.3 PWR register map

The following table summarizes the PWR register map and reset values.

#### Table 20. 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   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x004  | PWR_CSR  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

1. Available on STM32F070xB and STM32F030xC devices only.

Refer to Section 2.2 on page 37 for the register boundary addresses.
7 Reset and clock control (RCC)

7.1 Reset

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

7.1.1 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.

7.1.2 System reset

System reset sets each register to its reset value, unless otherwise specified in the register description.

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 event (WWDG reset)
3. Independent watchdog event (IWDG reset)
4. A software reset (SW reset) (see Software reset)
5. Low-power management reset (see Low-power management reset)
6. Option byte loader reset (see Option byte loader reset)
7. A power reset

The reset source can be identified by checking the reset flags in the Control/Status register, RCC_CSR (see Section 7.4.10: Control/status register (RCC_CSR)).

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.

The system reset signal provided to the device is output on the NRST pin. The pulse generator guarantees a minimum reset pulse duration of 20 µs for each internal reset source. In case of an external reset, the reset pulse is generated while the NRST pin is asserted low.
Software reset

The SYSRESETREQ bit in Arm® Cortex®-M0 Application Interrupt and Reset Control Register must be set to force a software reset on the device. Refer to the Cortex™-M0 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 Section 4: Option bytes.

Option byte loader reset

The option byte loader reset is generated when the OBL_LAUNCH bit (bit 13) is set in the FLASH_CR register. This bit is used to launch the option byte loading by software.

7.1.3 RTC domain reset

An RTC domain reset only affects the LSE oscillator, the RTC and the RCC RTC domain control register (RCC_BDCR). It is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RTC domain control register (RCC_BDCR).

2. A POR reset.
7.2 Clocks

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

- HSI 8 MHz RC oscillator clock
- HSE oscillator clock
- PLL clock

The devices have the following additional clock sources:

- 40 kHz low speed internal RC (LSI RC) which drives the independent watchdog and optionally the RTC used for Auto-wake-up from Stop/Standby mode.
- 32.768 kHz low speed external crystal (LSE crystal) which optionally drives the real-time clock (RTCCLOCK)
- 14 MHz high speed internal RC (HSI14) dedicated for ADC.

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

Several prescalers can be used to configure the frequency of the AHB and the APB domains. The AHB and the APB domains maximum frequency is 48 MHz.

All the peripheral clocks are derived from their bus clock (HCLK for AHB or PCLK for APB) except:

- The Flash memory programming interface clock (FLITFCLK) which is always the HSI clock.
- The option byte loader clock which is always the HSI clock
- The ADC clock which is derived (selected by software) from one of the two following sources:
  - dedicated HSI14 clock, to run always at the maximum sampling rate
  - APB clock (PCLK) divided by 2 or 4
- The USART1 clock which is derived (selected by software) from one of the four following sources:
  - system clock
  - HSI clock
  - LSE clock
  - APB clock (PCLK)
- The I2C1 clock which is derived (selected by software) from one of the two following sources:
  - system clock
  - HSI clock
- The USB clock which is derived (selected by software) from the following source:
  - PLL clock
- The RTC clock which is derived from the LSE, LSI or from the HSE clock divided by 32.
- The timer clock frequencies are automatically fixed by hardware. There are two cases:
  - if the APB prescaler is 1, the timer clock frequencies are set to the same frequency as that of the APB domain;
  - otherwise, they are set to twice (x2) the frequency of the APB domain.
- The IWDG clock which is always the LSI clock.
The RCC feeds the Cortex System Timer (SysTick) external clock with the AHB clock (HCLK) divided by 8. The SysTick can work either with this clock or directly with the Cortex clock (HCLK), configurable in the SysTick Control and Status Register.

Figure 10. Clock tree (STM32F030x4, STM32F030x6 and STM32F030x8 devices)

1. Applies to STM32F030x4/x6 devices.
2. Applies to STM32F030x8 devices.
FCLK acts as Arm® Cortex®-M0’s free-running clock. For more details refer to the Arm Cortex-M0 r0p0 technical reference manual (TRM).
7.2.1 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 12. HSE/ LSE clock sources**

<table>
<thead>
<tr>
<th>Clock source</th>
<th>Hardware configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>External clock</td>
<td><img src="MSv31915V1" alt="External clock Diagram" /></td>
</tr>
<tr>
<td>Crystal/Ceramic resonators</td>
<td><img src="MSv31916V1" alt="Crystal/Ceramic resonators Diagram" /></td>
</tr>
</tbody>
</table>
External crystal/ceramic resonator (HSE crystal)

The 4 to 32 MHz external oscillator has the advantage of producing a very accurate rate on the main clock.

The associated hardware configuration is shown in Figure 12. 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 HSE 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).

For code example refer to the Appendix section A.3.1: HSE start sequence.

Caution: To switch ON the HSE oscillator, 512 HSE clock pulses need to be seen by an internal stabilization counter after the HSEON bit is set. Even in the case that no crystal or resonator is connected to the device, excessive external noise on the OSC_IN pin may still lead the oscillator to start. Once the oscillator is started, it needs another 6 HSE clock pulses to complete a switching OFF sequence. If for any reason the oscillations are no more present on the OSC_IN pin, the oscillator cannot be switched OFF, locking the OSC pins from any other use and introducing unwanted power consumption. To avoid such situation, it is strongly recommended to always enable the Clock Security System (CSS) which is able to switch OFF the oscillator even in this case.

External source (HSE bypass)

In this mode, an external clock source must be provided. It can have a frequency of up to 32 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 ~40-60% duty cycle depending on the frequency (refer to the datasheet) has to drive the OSC_IN pin while the OSC_OUT pin can be used a GPIO. See Figure 12.

7.2.2 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 for 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 \( T_A=25^\circ C \).

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).
For more details on how to measure the HSI frequency variation refer to Section 7.2.12: Internal/external clock measurement with TIM14 on page 98.

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 7.2.7: Clock security system (CSS) on page 96.

Furthermore it is possible to drive the HSI clock to the MCO multiplexer. Then the clock could be driven to the Timer 14 giving the ability to the user to calibrate the oscillator.

### 7.2.3 PLL

The internal PLL can be used to multiply the HSI and the HSE output clock frequency. Refer to Figure 9: Simplified diagram of the reset circuit, Figure 12: HSE/ LSE clock sources and Clock control register (RCC_CR).

The PLL configuration (selection of the input clock, predivider and multiplication factor) must be done before enabling the PLL. Once the PLL is enabled, these parameters cannot be changed.

To modify the PLL configuration, proceed as follows:
1. Disable the PLL by setting PLLON to 0.
2. Wait until PLLRDY is cleared. The PLL is now fully stopped.
3. Change the desired parameter.
4. Enable the PLL again by setting PLLON to 1.
5. Wait until PLLRDY is set.

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

The PLL output frequency must be set in the range 16-48 MHz.

For code example refer to the Appendix section A.3.2: PLL configuration modification.

### 7.2.4 LSE clock

The LSE crystal is a 32.768 kHz Low Speed External crystal or ceramic resonator. It has the advantage of 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 RTC domain control register (RCC_BDCR). The crystal oscillator driving strength can be changed at runtime using the LSEDRV[1:0] bits in the RTC domain control register (RCC_BDCR) to obtain the best compromise between robustness and short start-up time on one side and low-power consumption on the other.

The LSERDY flag in the RTC domain control register (RCC_BDCR) indicates whether 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).
Caution: To switch ON the LSE oscillator, 4096 LSE clock pulses need to be seen by an internal stabilization counter after the LSEON bit is set. Even in the case that no crystal or resonator is connected to the device, excessive external noise on the OSC32_IN pin may still lead the oscillator to start. Once the oscillator is started, it needs another 6 LSE clock pulses to complete a switching OFF sequence. If for any reason the oscillations are no more present on the OSC_IN pin, the oscillator cannot be switched OFF, locking the OSC32 pins from any other use and introducing unwanted power consumption. The only way to recover such situation is to perform the RTC domain reset by software.

External source (LSE bypass)

In this mode, an external clock source must be provided. It can have a frequency of up to 1 MHz. You select this mode by setting the LSEBYP and LSEON bits in the RTC 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 can be used as GPIO. See Figure 12.

7.2.5 LSI clock

The LSI RC acts as a low-power clock source that can be kept running in Stop and Standby mode for the independent watchdog (IWDG) and RTC. The clock frequency is around 40 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 LSI 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).

7.2.6 System clock (SYSCLK) selection

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

- HSI oscillator
- HSE oscillator
- PLL

After a system reset, the HSI oscillator is selected as system clock. When a clock source is used directly or through the PLL as a 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 becomes ready. Status bits in the Clock control register (RCC_CR) indicate which clock(s) is (are) ready and which clock is currently used as a system clock.

7.2.7 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 clock, the HSE oscillator is automatically disabled, a clock failure event is sent to the break input of the advanced-control timers (TIM1) and general-purpose timers (TIM15, TIM16 and TIM17) and an interrupt is generated to inform the software about the failure (clock security system interrupt, or CSSI), allowing the MCU to
perform rescue operations. The CSSI is linked to the Arm® Cortex®-M0 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 is 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 HSE oscillator. If the HSE clock (divided or not) is the clock entry of the PLL used as system clock when the failure occurs, the PLL is disabled too.

7.2.8 ADC clock

The ADC clock selection is done inside the ADC_CFGR2 (refer to Section 12.10.5: ADC configuration register 2 (ADC_CFGR2) on page 220). It can be either the dedicated 14 MHz RC oscillator (HSI14) connected on the ADC asynchronous clock input or PCLK divided by 2 or 4. The 14 MHz RC oscillator can be configured by software either to be turned on/off ("auto-off mode") by the ADC interface or to be always enabled. The HSI 14 MHz RC oscillator cannot be turned on by ADC interface when the APB clock is selected as an ADC kernel clock.

7.2.9 RTC clock

The RTCCLK clock source can be either the HSE/32, LSE or LSI clocks. This is selected by programming the RTCSEL[1:0] bits in the RTC domain control register (RCC_BDCR). This selection cannot be modified without resetting the RTC domain. The system must be always configured in a way that the PCLK frequency is greater then or equal to the RTCCCLK frequency for proper operation of the RTC.

7.2.10 Independent 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.

7.2.11 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 the following clock signals can be selected as the MCO clock:

- HSI14
- SYSCLK
- HSI
- HSE
- PLL clock divided by 2 or direct (direct connection is not available on STM32F030x8 devices)
- LSE
- LSI
The selection is controlled by the MCO[3:0] bits of the Clock configuration register (RCC_CFGR).

For code example refer to the Appendix section A.3.3: MCO selection.

On STM32F030x4, STM32F030x6, STM32F070xB, and STM32F030xC devices, the additional bit PLLNODIV of this register controls the divider bypass for a PLL clock input to MCO. The MCO frequency can be reduced by a configurable binary divider, controlled by the MCOPRE[2..0] bits of the Clock configuration register (RCC_CFGR).

### 7.2.12 Internal/external clock measurement with TIM14

It is possible to indirectly measure the frequency of all on-board clock sources by mean of the TIM14 channel 1 input capture. As represented on Figure 13.

**Figure 13. Frequency measurement with TIM14 in capture mode**

The input capture channel of the Timer 14 can be a GPIO line or an internal clock of the MCU. This selection is performed through the TI1_RMP[1:0] bits in the TIM14_OR register. The possibilities available are the following ones.

- TIM14 Channel1 is connected to the GPIO. Refer to the alternate function mapping in the device datasheets.
- TIM14 Channel1 is connected to the RTCCLK.
- TIM14 Channel1 is connected to the HSE/32 Clock.
- TIM14 Channel1 is connected to the microcontroller clock output (MCO). Refer to Section 7.2.11: Clock-out capability for MCO clock configuration.

For code example refer to the Appendix section A.3.4: Clock measurement configuration with TIM14.

**Calibration of the HSI**

The primary purpose of connecting the LSE, through the MCO multiplexer, to the channel 1 input capture is to be able to precisely measure the HSI system clocks (for this, the HSI should be used as the system clock source). The number of HSI clock counts between consecutive edges of the LSE signal provides a measure of the internal clock period. Taking advantage of the high precision of LSE crystals (typically a few tens of ppm), it is possible to determine the internal clock frequency with the same resolution, and trim the source to compensate for manufacturing-process- and/or temperature- and voltage-related frequency deviations.

The HSI oscillator has dedicated user-accessible calibration bits for this purpose.
The basic concept consists in providing a relative measurement (e.g. the HSI/LSE ratio): the precision is therefore closely related to the ratio between the two clock sources. The higher the ratio is, the better the measurement is.

If LSE is not available, HSE/32 is the better option in order to reach the most precise calibration possible.

### Calibration of the LSI

The calibration of the LSI will follow the same pattern that for the HSI, but changing the reference clock. It is necessary to connect LSI clock to the channel 1 input capture of the TIM14. Then define the HSE as system clock source, the number of its clock counts between consecutive edges of the LSI signal provides a measure of the internal low speed clock period.

The basic concept consists in providing a relative measurement (e.g. the HSE/LSI ratio): the precision is therefore closely related to the ratio between the two clock sources. The higher the ratio is, the better the measurement is.

### Calibration of the HSI14

For the HSI14, because of its high frequency, it is not possible to have a precise resolution. However a solution could be to clock Timer 14 with HSE through PLL to reach 48 MHz, and to use the input capture line with the HSI14 and the capture prescaler defined to the higher value. In that configuration, we got a ratio of 27 events. It is still a bit low to have an accurate calibration. In order to increase the measure accuracy, it is advised to count the HSI periods after multiple cycles of Timer 14. Using polling to treat the capture event is necessary in this case.

### 7.3 Low-power modes

APB peripheral clocks and DMA clock can be disabled by software.

Sleep mode stops the CPU clock. The memory interface clocks (Flash and RAM interfaces) can be stopped by software during sleep mode. The AHB to APB bridge clocks are disabled by hardware during Sleep mode when all the clocks of the peripherals connected to them are disabled.

Stop mode stops all the clocks in the core supply domain and disables the PLL and the HSI, HSI14 and HSE oscillators.

Standby mode stops all the clocks in the core supply domain and disables the PLL and the HSI, HSI14 and HSE oscillators.

The CPU’s deepsleep mode can be overridden for debugging by setting the DBG_STOP or DBG_STANDBY bits in the DBGMCU_CR register.

When waking up from deepsleep after an interrupt (Stop mode) or reset (Standby mode), the HSI oscillator is selected as system clock.

If a Flash programming operation is on going, deepsleep mode entry is delayed until the Flash interface access is finished. If an access to the APB domain is ongoing, deepsleep mode entry is delayed until the APB access is finished.
7.4 RCC registers

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

7.4.1 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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
</table>
| 31:26 | Reserved, must be kept at reset value. |}

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, must be kept at reset value.

Bit 19 CSSON: Clock security system enable
Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if a HSE clock failure is detected.
0: Clock security system disabled (clock detector OFF).
1: Clock security system enabled (clock detector ON if the HSE is ready, OFF if not).

Bit 18 HSEBYP: HSE crystal oscillator bypass
Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit set, to be used by the device. The HSEBYP bit can be written only if the HSE oscillator is disabled.
0: HSE crystal oscillator not bypassed
1: HSE crystal oscillator bypassed with external clock
Bit 17 **HSERDY**: HSE clock ready flag
Set by hardware to indicate that the HSE oscillator is stable. This bit needs 6 cycles of the HSE oscillator clock to fall down after HSEON reset.
- 0: HSE oscillator not ready
- 1: HSE oscillator ready

Bit 16 **HSEON**: HSE clock enable
Set and cleared by software.
Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.
- 0: HSE oscillator OFF
- 1: HSE oscillator ON

Bits 15:8 **HSICAL[7:0]**: HSI clock calibration
These bits are initialized automatically at startup. They are adjusted by SW through the HSITRIM setting.

Bits 7:3 **HSITRIM[4:0]**: HSI 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 HSI.
The default value is 16, which, when added to the HSICAL value, should trim the HSI to 8 MHz ± 1%. The trimming step is around 40 kHz between two consecutive HSICAL steps.
*Note: Increased value in the register results to higher clock frequency.*

Bit 2 Reserved, must be kept at reset value.

Bit 1 **HSIRDY**: HSI clock ready flag
Set by hardware to indicate that HSI oscillator is stable. After the HSION bit is cleared, HSIRDY goes low after 6 HSI oscillator clock cycles.
- 0: HSI oscillator not ready
- 1: HSI oscillator ready

Bit 0 **HSION**: HSI clock enable
Set and cleared by software.
Set by hardware to force the HSI oscillator ON when leaving Stop or Standby mode or in case of failure of the HSE crystal oscillator used directly or indirectly as system clock. This bit cannot be reset if the HSI is used directly or indirectly as system clock or is selected to become the system clock.
- 0: HSI oscillator OFF
- 1: HSI oscillator ON

### 7.4.2 Clock configuration register (RCC_CFRG)

**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.
### Reset and clock control (RCC)

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>PLLNODIV: PLL clock not divided for MCO (not available on STM32F030x8 devices)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This bit is set and cleared by software. It switches off divider by 2 for PLL connection to MCO.</td>
</tr>
<tr>
<td></td>
<td>0: PLL is divided by 2 for MCO</td>
</tr>
<tr>
<td></td>
<td>1: PLL is not divided for MCO</td>
</tr>
</tbody>
</table>

### Bits 30:28 MCOPRE[2:0]: Microcontroller clock output prescaler (not available on STM32F030x8 devices)

These bits are set and cleared by software to select the MCO prescaler division factor. To avoid glitches, it is highly recommended to change this prescaler only when the MCO output is disabled.

- 000: MCO is divided by 1
- 001: MCO is divided by 2
- 010: MCO is divided by 4
- .....  
- 111: MCO is divided by 128

### Bits 27:22 MCO[3:0]: Microcontroller clock output

Set and cleared by software.

- 0000: MCO output disabled, no clock on MCO
- 0001: Internal RC 14 MHz (HSI14) oscillator clock selected
- 0010: Internal low speed (LSI) oscillator clock selected
- 0011: External low speed (LSE) oscillator clock selected
- 0100: System clock selected
- 0101: Internal RC 8 MHz (HSI) oscillator clock selected
- 0110: External 4-32 MHz (HSE) oscillator clock selected
- 0111: PLL clock selected (divided by 1 or 2, depending on PLLNODIV)

1xxx: Reserved, must be kept at reset value.

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

### Bits 23:22 Reserved, must be kept at reset value.
Bits 21:18 **PLL_MUL[3:0]**: 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 48 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

Bit 17 **PLL_XTPRE**: HSE divider for PLL input clock
This bit is the same bit as bit PREDIV[0] from RCC_CFGR2. Refer to RCC_CFGR2 PREDIV bits description for its meaning.

Bits 16:15 **PLL_SRC[1:0]**: PLL input clock source
Set and cleared by software to select PLL or PREDIV clock source. These bits can be written only when PLL is disabled.
- 00: HSI/2 selected as PLL input clock (PREDIV forced to divide by 2 on STM32F070xx and STM32F030xC devices)
- 01: HSI/PREDIV selected as PLL input clock
- 10: HSE/PREDIV selected as PLL input clock
- 11: Reserved
Bit PLL_SRC[0] is available only on STM32F070xx and STM32F030xC devices, otherwise it is reserved (with value zero).

Bit 14 **ADCPRE**: ADC prescaler
Obsolete setting. Proper ADC clock selection is done inside the ADC_CFGR2 (refer to Section 12.10.5: ADC configuration register 2 (ADC_CFGR2) on page 220).

Bits 13:11 Reserved, must be kept at reset value.

Bits 10:8 **PPRE[2:0]**: PCLK prescaler
Set and cleared by software to control the division factor of the APB clock (PCLK).
- 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[3:0]**: HCLK 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

Bits 3:2 **SWS[1:0]**: 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: Reserved, must be kept at reset value.

Bits 1:0 **SW[1:0]**: System clock switch
Set and cleared by software to select SYSCLK source.

- Cleared 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: Reserved, must be kept at reset value.

### 7.4.3 Clock interrupt register (RCC_CIR)

**Address offset:** 0x08

**Reset value:** 0x0000 0000

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

<table>
<thead>
<tr>
<th>Bits 31:24</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 23 <strong>CSSC</strong>: Clock security system interrupt clear</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This bit is set by software to clear the CSSF flag.</td>
</tr>
<tr>
<td></td>
<td>0: No effect</td>
</tr>
<tr>
<td></td>
<td>1: Clear CSSF flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits 23:16</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 15:14</td>
<td>HSI14_RDYF</td>
</tr>
<tr>
<td>Bits 13:12</td>
<td>PLL_RDYF</td>
</tr>
<tr>
<td>Bits 11:10</td>
<td>HSE_RDYF</td>
</tr>
<tr>
<td>Bits 9:8</td>
<td>LSE_RDYF</td>
</tr>
<tr>
<td>Bits 7:6</td>
<td>LSI_RDYF</td>
</tr>
<tr>
<td>Bits 5:4</td>
<td>CSSF</td>
</tr>
<tr>
<td>Bits 3:2</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>Bits 1:0</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>
Bit 22
Reserved, must be kept at reset value.

Bit 21 **HSI14RDYC**: HSI14 ready interrupt clear
This bit is set by software to clear the HSI14RDYF flag.
  0: No effect
  1: Clear HSI14RDYF flag

Bit 20 **PLLRDYC**: PLL ready interrupt clear
This bit is set by software to clear the PLLRDYF flag.
  0: No effect
  1: Clear PLLRDYF flag

Bit 19 **HSERDYC**: HSE ready interrupt clear
This bit is set by software to clear the HSERDYF flag.
  0: No effect
  1: Clear HSERDYF flag

Bit 18 **HSIRDYC**: HSI ready interrupt clear
This bit is set by software to clear the HSIRDYF flag.
  0: No effect
  1: Clear HSIRDYF flag

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 15:14 Reserved, must be kept at reset value.

Bit 13 **HSI14RDYIE**: HSI14 ready interrupt enable
Set and cleared by software to enable/disable interrupt caused by the HSI14 oscillator stabilization.
  0: HSI14 ready interrupt disabled
  1: HSI14 ready interrupt enabled

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 HSE 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 HSI 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 LSE 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 the LSI 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 HSE 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

Bit 6 Reserved, must be kept at reset value.

Bit 5 **HSI14RDYF**: HSI14 ready interrupt flag
Set by hardware when the HSI14 becomes stable and HSI14RDYDIE is set in a response to setting the HSI14ON bit in Clock control register 2 (RCC_CR2). When HSI14ON is not set but the HSI14 oscillator is enabled by the peripheral through a clock request, this bit is not set and no interrupt is generated.
Cleared by software setting the HSI14RDYC bit.
0: No clock ready interrupt caused by the HSI14 oscillator
1: Clock ready interrupt caused by the HSI14 oscillator

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

Bit 3 **HSERDYF**: HSE ready interrupt flag
Set by hardware when the HSE clock becomes stable and HSERDYDIE is set.
Cleared by software setting the HSERDYC bit.
0: No clock ready interrupt caused by the HSE oscillator
1: Clock ready interrupt caused by the HSE oscillator

Bit 2 **HSIRDYF**: HSI ready interrupt flag
Set by hardware when the HSI clock becomes stable and HSIRDYDIE is set in a response to setting the HSION (refer to Clock control register (RCC_CR)). When HSION is not set but the HSI oscillator is enabled by the peripheral through a clock request, this bit is not set and no interrupt is generated.
Cleared by software setting the HSIRDYC bit.
0: No clock ready interrupt caused by the HSI oscillator
1: Clock ready interrupt caused by the HSI oscillator
Bit 1 **LSERDYF**: LSE ready interrupt flag
Set by hardware when the LSE clock becomes stable and LSERDYDIE is set.
Cleared by software setting the LSERDYC bit.
0: No clock ready interrupt caused by the LSE oscillator
1: Clock ready interrupt caused by the LSE oscillator

Bit 0 **LSIRDYF**: LSI ready interrupt flag
Set by hardware when the LSI clock becomes stable and LSIRDYDIE is set.
Cleared by software setting the LSIRDYC bit.
0: No clock ready interrupt caused by the LSI oscillator
1: Clock ready interrupt caused by the LSI oscillator

### 7.4.4 APB peripheral reset register 2 (RCC_APB2RSTR)

Address offset: 0x0C
Reset value: 0x00000 0000
Access: no wait state, word, half-word and byte access

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>SPI1RST: SPI1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SPI1RST: SPI1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SPI1RST: SPI1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>USART1RST: USART1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SPI1RST: SPI1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TIM1RST: TIM1 reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ADCRST: ADC reset</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>DBGMCURST: Debug MCU reset</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:23 Reserved, must be kept at reset value.

Bit 22 **DBGMCURST**: Debug MCU reset
Set and cleared by software.
0: No effect
1: Reset Debug MCU

Bits 21:19 Reserved, must be kept at reset value.

Bit 18 **TIM17RST**: TIM17 timer reset
Set and cleared by software.
0: No effect
1: Reset TIM17 timer

Bit 17 **TIM16RST**: TIM16 timer reset
Set and cleared by software.
0: No effect
1: Reset TIM16 timer

Bit 16 **TIM15RST**: TIM15 timer reset
Set and cleared by software.
0: No effect
1: Reset TIM15 timer

Bit 15 Reserved, must be kept at reset value.
Bit 14 **USART1RST**: USART1 reset  
Set and cleared by software.  
0: No effect  
1: Reset USART1

Bit 13 Reserved, must be kept at reset value.

Bit 12 **SPI1RST**: SPI1 reset  
Set and cleared by software.  
0: No effect  
1: Reset SPI1

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

Bit 10 Reserved, must be kept at reset value.

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

Bits 8:6 Reserved, must be kept at reset value.

Bit 5 **USART6RST**: USART6 reset  
Set and cleared by software  
0: No effect  
1: Reset USART6

Bits 4:1 Reserved, must be kept at reset value.

Bit 0 **SYSCFG RST**: SYSCFG reset  
Set and cleared by software.  
0: No effect  
1: Reset SYSCFG

### 7.4.5 APB peripheral reset register 1 (RCC_APB1RSTR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access
Bits 31:29  Reserved, must be kept at reset value.

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

Bits 27:24  Reserved, must be kept at reset value.

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

Bit 22  **I2C2RST**: I2C2 reset
Set and cleared by software.
0: No effect
1: Reset I2C2

Bit 21  **I2C1RST**: I2C1 reset
Set and cleared by software.
0: No effect
1: Reset I2C1

Bit 20  **USART5RST**: USART5 reset
Set and cleared by software.
0: No effect
1: Reset USART4

Bit 19  **USART4RST**: USART4 reset
Set and cleared by software.
0: No effect
1: Reset USART4

Bit 18  **USART3RST**: USART3 reset
Set and cleared by software.
0: No effect
1: Reset USART3

Bit 17  **USART2RST**: USART2 reset
Set and cleared by software.
0: No effect
1: Reset USART2

Bits 16:15  Reserved, must be kept at reset value.

Bit 14  **SPI2RST**: SPI2 reset
Set and cleared by software.
0: No effect
1: Reset SPI2

Bits 13:12  Reserved, must be kept at reset value.

Bit 11  **WWDGRST**: Window watchdog reset
Set and cleared by software.
0: No effect
1: Reset window watchdog
7.4.6  

**AHB peripheral clock enable register (RCC_AHBENR)**

Address offset: 0x14  
Reset value: 0x0000 0014  
Access: no wait state, word, half-word and byte access

*Note:* When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

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

Bits 31:23  Reserved, must be kept at reset value.

Bit 22  **GPIOFEN:** 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 21  Reserved, must be kept at reset value.
7.4.7 APB peripheral clock enable register 2 (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 APB domain is on going. In this case, wait states are inserted until the access to APB peripheral is finished.
**Note:** When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:23 Reserved, must be kept at reset value.

Bit 22 **DBGMCUEN** MCU debug module clock enable
- Set and reset by software.
  - 0: MCU debug module clock disabled
  - 1: MCU debug module enabled

Bits 21:19 Reserved, must be kept at reset value.

Bit 18 **TIM17EN**: TIM17 timer clock enable
- Set and cleared by software.
  - 0: TIM17 timer clock disabled
  - 1: TIM17 timer clock enabled

Bit 17 **TIM16EN**: TIM16 timer clock enable
- Set and cleared by software.
  - 0: TIM16 timer clock disabled
  - 1: TIM16 timer clock enabled

Bit 16 **TIM15EN**: TIM15 timer clock enable
- Set and cleared by software.
  - 0: TIM15 timer clock disabled
  - 1: TIM15 timer clock enabled

Bit 15 Reserved, must be kept at reset value.

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

Bit 13 Reserved, must be kept at reset value.

Bit 12 **SPI1EN**: SPI1 clock enable
- Set and cleared by software.
  - 0: SPI1 clock disabled
  - 1: SPI1 clock enabled

Bit 11 **TIM1EN**: TIM1 timer clock enable
- Set and cleared by software.
  - 0: TIM1 timer clock disabled
  - 1: TIM1P timer clock enabled
Bit 10 Reserved, must be kept at reset value.

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

Bits 8:6 Reserved, must be kept at reset value.

Bit 5 USART6EN: USART6 clock enable
Set and cleared by software.
0: USART6 clock disabled
1: USART6 clock enabled

Bits 4:1 Reserved, must be kept at reset value.

Bit 0 SYSCFGEN: SYSCFG clock enable
Set and cleared by software.
0: SYSCFG clock disabled
1: SYSCFG clock enabled

### 7.4.8 APB peripheral clock enable register 1 (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 APB domain is going on. In this case, wait states are inserted until this access to APB peripheral is finished.

*Note:* When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
<th>Access</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Res.</td>
<td>rw</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>30</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>USB EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>I2C2 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>I2C1 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>USART5 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>USART4 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>USART3 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>USART2 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PWREN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SPI2 EN</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Res.</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:29 Reserved, must be kept at reset value.

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

Bits 27:24 Reserved, must be kept at reset value.
Bit 23  **USBEN**: USB interface clock enable  
Set and cleared by software.  
0: USB interface clock disabled  
1: USB interface clock enabled

Bit 22  **I2C2EN**: I2C2 clock enable  
Set and cleared by software.  
0: I2C2 clock disabled  
1: I2C2 clock enabled

Bit 21  **I2C1EN**: I2C1 clock enable  
Set and cleared by software.  
0: I2C1 clock disabled  
1: I2C1 clock enabled

Bit 20  **USART5EN**: USART5 clock enable  
Set and cleared by software.  
0: USART5 clock disabled  
1: USART5 clock enabled

Bit 19  **USART4EN**: USART4 clock enable  
Set and cleared by software.  
0: USART4 clock disabled  
1: USART4 clock enabled

Bit 18  **USART3EN**: USART3 clock enable  
Set and cleared by software.  
0: USART3 clock disabled  
1: USART3 clock enabled

Bit 17  **USART2EN**: USART2 clock enable  
Set and cleared by software.  
0: USART2 clock disabled  
1: USART2 clock enabled

Bits 16:15  Reserved, must be kept at reset value.

Bit 14  **SPI2EN**: SPI2 clock enable  
Set and cleared by software.  
0: SPI2 clock disabled  
1: SPI2 clock enabled

Bits 13:12  Reserved, must be kept at reset value.

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

Bits 10:9  Reserved, must be kept at reset value.

Bit 8  **TIM14EN**: TIM14 timer clock enable  
Set and cleared by software.  
0: TIM14 clock disabled  
1: TIM14 clock enabled

Bits 7:6  Reserved, must be kept at reset value.
Bit 5 **TIM7EN**: TIM7 timer clock enable (not available on STM32F070x6, nor STM32F030x4/6/8/C devices).
   Set and cleared by software.
   0: TIM7 clock disabled
   1: TIM7 clock enabled

Bit 4 **TIM6EN**: TIM6 timer clock enable
   Set and cleared by software.
   0: TIM6 clock disabled
   1: TIM6 clock enabled

Bits 3:2 Reserved, must be kept at reset value.

Bit 1 **TIM3EN**: TIM3 timer clock enable
   Set and cleared by software.
   0: TIM3 clock disabled
   1: TIM3 clock enabled

Bit 0 Reserved, must be kept at reset value.

### 7.4.9 RTC domain control register (RCC_BDCR)

Address offset: 0x20
Reset value: 0x0000 0018, reset by RTC 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:** The LSEON, LSEBYP, RTCSEL and RTCCEN bits of the RTC domain control register (RCC_BDCR) are in the RTC 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 they can be modified. Refer to Section 6.1.2: Voltage regulator for further information. These bits are only reset after a RTC domain reset (see Section 7.1.3: RTC domain reset). Any internal or external reset does not have any effect on these bits.

<table>
<thead>
<tr>
<th>Bit 31-17</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RTC</strong></td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td><strong>BDRST</strong></td>
<td>RTC domain software reset</td>
</tr>
</tbody>
</table>

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **BDRST**: RTC domain software reset
   Set and cleared by software.
   0: Reset not activated
   1: Resets the entire RTC domain
Bit 15  **RTCEN**: RTC clock enable
    Set and cleared by software.
    0: RTC clock disabled
    1: RTC clock enabled

Bits 14:10  Reserved, must be kept at reset value.

Bits 9:8  **RTCSSEL[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 RTC 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 32 used as RTC clock

Bits 7:5  Reserved, must be kept at reset value.

Bits 4:3  **LSEDRV**: LSE oscillator drive capability
    Set and reset by software to modulate the LSE oscillator’s drive capability. A reset of the RTC
    domain restores the default value.
    00: ‘Xtal mode’ low drive capability
    01: ‘Xtal mode’ medium-high drive capability
    10: ‘Xtal mode’ medium-low drive capability
    11: ‘Xtal mode’ high drive capability (reset value)

*Note*: The oscillator is in Xtal mode when it is not in bypass mode.

Bit 2  **LSEBYP**: LSE 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**: LSE 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: LSE oscillator not ready
    1: LSE oscillator ready

Bit 0  **LSEON**: LSE oscillator enable
    Set and cleared by software.
    0: LSE oscillator OFF
    1: LSE oscillator ON

### 7.4.10 Control/status register (RCC_CSR)

Address: 0x24

Reset value: 0xXXX0 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.
**Bit 31** **LPWRSTF**: 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 refer to [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 VDD 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** **OBLRSTF**: Option byte loader reset flag
- Set by hardware when a reset from the OBL occurs.
- Cleared by writing to the RMVF bit.
  - 0: No reset from OBL occurred
  - 1: Reset from OBL occurred
Bit 24 **RMVF**: Remove reset flag
Set by software to clear the reset flags including RMVF.
0: No effect
1: Clear the reset flags

Bit 23 **V18PWRRSTF**: Reset flag of the 1.8 V domain.
Set by hardware when a POR/PDR of the 1.8 V domain occurred.
Cleared by writing to the RMVF bit.
0: No POR/PDR reset of the 1.8 V domain occurred
1: POR/PDR reset of the 1.8 V domain occurred

Bits 22:2 Reserved, must be kept at reset value.

Bit 1 **LSIRDY**: LSI oscillator ready
Set and cleared by hardware to indicate when the LSI oscillator is stable. After the LSION bit is cleared, LSIRDY goes low after 3 LSI oscillator clock cycles.
0: LSI oscillator not ready
1: LSI oscillator ready

Bit 0 **LSION**: LSI oscillator enable
Set and cleared by software.
0: LSI oscillator OFF
1: LSI oscillator ON

### 7.4.11 AHB peripheral reset register (RCC_AHBRSTR)

**Address**: 0x28

**Reset value**: 0x0000 0000

**Access**: no wait states, word, half-word and byte access

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 31:23 Reserved, must be kept at reset value.

Bit 22 **GPIOFRST**: I/O port F reset
Set and cleared by software.
0: No effect
1: Reset I/O port F

Bit 21 Reserved, must be kept at reset value.
Bit 20 **GPIODRST**: I/O port D reset
   Set and cleared by software.
   0: No effect
   1: Reset I/O port D

Bit 19 **GPIOCRST**: I/O port C reset
   Set and cleared by software.
   0: No effect
   1: Reset I/O port C

Bit 18 **GPIOBRST**: I/O port B reset
   Set and cleared by software.
   0: No effect
   1: Reset I/O port B

Bit 17 **GPIOARST**: I/O port A reset
   Set and cleared by software.
   0: No effect
   1: Reset I/O port A

Bits 16:0 Reserved, must be kept at reset value.

### 7.4.12 Clock configuration register 2 (RCC_CFGR2)

Address: 0x2C
Reset value: 0x0000 0000
Access: no wait states, word, half-word and byte access

```
<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Bits 31:4 Reserved, must be kept at reset value.
Bits 3:0 **PREDIV[3:0]** PREDIV division factor

These bits are set and cleared by software to select PREDIV division factor. They can be written only when the PLL is disabled.

*Note:* Bit 0 is the same bit as bit 17 in Clock configuration register (RCC_CFGR), so modifying bit 17 in Clock configuration register (RCC_CFGR) also modifies bit 0 in Clock configuration register 2 (RCC_CFGR2) (for compatibility with other STM32 products)

- 0000: PREDIV input clock not divided
- 0001: PREDIV input clock divided by 2
- 0010: PREDIV input clock divided by 3
- 0011: PREDIV input clock divided by 4
- 0100: PREDIV input clock divided by 5
- 0101: PREDIV input clock divided by 6
- 0110: PREDIV input clock divided by 7
- 0111: PREDIV input clock divided by 8
- 1000: PREDIV input clock divided by 9
- 1001: PREDIV input clock divided by 10
- 1010: PREDIV input clock divided by 11
- 1011: PREDIV input clock divided by 12
- 1100: PREDIV input clock divided by 13
- 1101: PREDIV input clock divided by 14
- 1110: PREDIV input clock divided by 15
- 1111: PREDIV input clock divided by 16

### 7.4.13 Clock configuration register 3 (RCC_CFGR3)

**Address:** 0x30  
**Reset value:** 0x0000 0000  
**Access:** no wait states, word, half-word and byte access

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

**Bits 31:9** Reserved, must be kept at reset value.

**Bit 8 ADCSW:** ADC clock source selection

Obsolete setting. To be kept at reset value, connecting the HSI14 clock to the ADC asynchronous clock input. Proper ADC clock selection is done inside the ADC_CFRG2 (refer to Section 12.10.5: ADC configuration register 2 (ADC_CFRG2) on page 220).

**Bit 7 USBSW:** USB clock source selection

This bit is set and cleared by software to select the USB clock source.  
0: USB clock disabled (default)  
1: PLL clock (PLLCLK) selected as USB clock
7.4.14 Clock control register 2 (RCC_CR2)

Address: 0x34

Reset value: 0xXX00 XX80, where X is undefined.

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

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:8 HSI14CAL[7:0]: HSI14 clock calibration
These bits are initialized automatically at startup.

Bits 7:3 HSI14TRIM[4:0]: HSI14 clock trimming
These bits provide an additional user-programmable trimming value that is added to the HSI14CAL[7:0] bits. It can be programmed to adjust to variations in voltage and temperature that influence the frequency of the HSI14.

The default value is 16, which, when added to the HSI14CAL value, should trim the HSI14 to 14 MHz ± 1%. The trimming step is around 50 kHz between two consecutive HSI14CAL steps.

Bit 2 HSI14DIS HSI14 clock request from ADC disable
Set and cleared by software.

When set this bit prevents the ADC interface from enabling the HSI14 oscillator.

0: ADC interface can turn on the HSI14 oscillator
1: ADC interface can not turn on the HSI14 oscillator
Bit 1  **HSI14RDY**: HSI14 clock ready flag  
Set by hardware to indicate that HSI14 oscillator is stable. After the HSI14ON bit is cleared, HSI14RDY goes low after 6 HSI14 oscillator clock cycles. When HSI14ON is not set but the HSI14 oscillator is enabled by the peripheral through a clock request, this bit is not set.  
0: HSI14 oscillator not ready  
1: HSI14 oscillator ready  

Bit 0  **HSI14ON**: HSI14 clock enable  
Set and cleared by software. When the HSI14 oscillator is enabled by the peripheral through a clock request, this bit is not set and resetting it does not stop the HSI14 oscillator.  
0: HSI14 oscillator OFF  
1: HSI14 oscillator ON
7.4.15 RCC register map

The following table gives the RCC register map and the 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 | 0 |
|--------|--------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0x00   | RCC_CR             | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  |
|        | Reset value        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  |
| 0x04   | RCC_CFGR           | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x08   | RCC_CIR            | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x0C   | RCC_APB2RSTR       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x10   | RCC_APB1RSTR       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x14   | RCC_AHBENR          | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x18   | RCC_APB2ENR        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x1C   | RCC_APB1ENR        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x20   | RCC_BDCR           | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |
| 0x24   | RCC_CSR            | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  

The following table gives the RCC register map and the reset values.
Refer to Section 2.2 on page 37 for the register boundary addresses.
8 General-purpose I/Os (GPIO)

8.1 Introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR) and a 32-bit set/reset register (GPIOx_BSRR). Ports A and B also have a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection registers (GPIOx_AFRH and GPIOx_AFRL).

On STM32F030xB and STM32F030xC devices, also ports C and D have two 32-bit alternate function selection registers (GPIOx_AFRH and GPIOx_AFRL).

8.2 GPIO main features

- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIOx_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIOx_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIOx_BSRR) for bitwise write access to GPIOx_ODR
- Locking mechanism (GPIOx_LCKR) provided to freeze the port A or B I/O port configuration.
- Analog function
- Alternate function selection registers (at most 16 AFs possible per I/O)
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

8.3 GPIO functional description

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

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

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words, half-words or bytes. The purpose of the GPIOx_BSRR register is
to allow atomic read/modify accesses to any of the GPIOx_ODR registers. In this way, there
is no risk of an IRQ occurring between the read and the modify access.

*Figure 14* shows the basic structures of a standard I/O port bit. *Table 22* gives the possible
port bit configurations.

*Figure 14. Basic structure of an I/O port bit*

![Diagram of I/O port bit structure]

*Table 22. Port bit configuration table*(1)

<table>
<thead>
<tr>
<th>MODER(i) [1:0]</th>
<th>OTYPER(i)</th>
<th>OSPEEDR(i) [1:0]</th>
<th>PUPDR(i) [1:0]</th>
<th>I/O configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>GP output PP</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>01</td>
<td>00</td>
<td>GP output PP + PU</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>00</td>
<td>GP output PP + PD</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>11</td>
<td>00</td>
<td>GP output OD</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>00</td>
<td>00</td>
<td>GP output OD + PU</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>01</td>
<td>00</td>
<td>GP output OD + PD</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>11</td>
<td>00</td>
<td>Reserved (GP output OD)</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>AF PP</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>01</td>
<td>00</td>
<td>AF PP + PU</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>00</td>
<td>AF PP + PD</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>11</td>
<td>00</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>00</td>
<td>00</td>
<td>AF OD</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>01</td>
<td>00</td>
<td>AF OD + PU</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>10</td>
<td>00</td>
<td>AF OD + PD</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>11</td>
<td>00</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
8.3.1 General-purpose I/O (GPIO)

During and just after reset, the alternate functions are not active and most of the I/O ports are configured in input floating mode.

The debug pins are in AF pull-up/pull-down after reset:
- PA14: SWCLK in pull-down
- PA13: SWDIO in pull-up

When the pin is 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 low level is driven, high level is HI-Z).

The input data register (GPIOx_IDR) captures the data present on the I/O pin at every AHB clock cycle.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not depending on the value in the GPIOx_PUPDR register.

8.3.2 I/O pin alternate function multiplexer and mapping

The device I/O pins are connected to on-board peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an I/O pin at a time. In this way, there can be no conflict between peripherals available on the same I/O pin.

Each I/O pin has a multiplexer with up to sixteen alternate function inputs (AF0 to AF15) that can be configured through the GPIOx_AFRL (for pin 0 to 7) and GPIOx_AFRH (for pin 8 to 15) registers:
- After reset the multiplexer selection is alternate function 0 (AF0). The I/Os are configured in alternate function mode through GPIOx_MODER register.
- The specific alternate function assignments for each pin are detailed in the device datasheet.

In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped onto different I/O pins to optimize the number of peripherals available in smaller packages.
To use an I/O in a given configuration, the user has to proceed as follows:

- **Debug function**: after each device reset these pins are assigned as alternate function pins immediately usable by the debugger host.
- **GPIO**: configure the desired I/O as output, input or analog in the GPIOx_MODER register.
- **Peripheral alternate function**:
  - Connect the I/O to the desired AFx in one of the GPIOx_AFRL or GPIOx_AFRH register.
  - Select the type, pull-up/pull-down and output speed via the GPIOx_OTYPER, GPIOx_PUPDR and GPIOx_OSPEEDER registers, respectively.
  - Configure the desired I/O as an alternate function in the GPIOx_MODER register.
- **Additional functions**:
  - ADC connection can be enabled in ADC registers regardless the configured GPIO mode. When ADC uses a GPIO, it is recommended to configure the GPIO in analog mode, through the GPIOx_MODER register.
  - For the additional functions like RTC, WKUPx and oscillators, configure the required function in the related RTC, PWR and RCC registers. These functions have priority over the configuration in the standard GPIO registers.

Refer to the “Alternate function mapping” table in the device datasheet for the detailed mapping of the alternate function I/O pins.

### 8.3.3 I/O port control registers

Each of the GPIO ports has four 32-bit memory-mapped control registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR) to configure up to 16 I/Os. The GPIOx_MODER register is used to select the I/O mode (input, output, AF, analog). The GPIOx_OTYPER and GPIOx_OSPEEDR registers are used to select the output type (push-pull or open-drain) and speed. The GPIOx_PUPDR register is used to select the pull-up/pull-down whatever the I/O direction.

### 8.3.4 I/O port data registers

Each GPIO has two 16-bit memory-mapped data registers: input and output data registers (GPIOx_IDR and GPIOx_ODR). GPIOx_ODR stores the data to be output, it is read/write accessible. The data input through the I/O are stored into the input data register (GPIOx_IDR), a read-only register.

See Section 8.4.5: GPIO port input data register (GPIOx_IDR) \((x = A to D, F)\) and Section 8.4.6: GPIO port output data register (GPIOx_ODR) \((x = A to D, F)\) for the register descriptions.

### 8.3.5 I/O data bitwise handling

The bit set reset register (GPIOx_BSRR) is a 32-bit register which allows the application to set and reset each individual bit in the output data register (GPIOx_ODR). The bit set reset register has twice the size of GPIOx_ODR.

To each bit in GPIOx_ODR, correspond two control bits in GPIOx_BSRR: BS(i) and BR(i). When written to 1, bit BS(i) sets the corresponding ODR(i) bit. When written to 1, bit BR(i) resets the ODR(i) corresponding bit.
Writing any bit to 0 in GPIOx_BSRR does not have any effect on the corresponding bit in GPIOx_ODR. If there is an attempt to both set and reset a bit in GPIOx_BSRR, the set action takes priority.

Using the GPIOx_BSRR register to change the values of individual bits in GPIOx_ODR is a “one-shot” effect that does not lock the GPIOx_ODR bits. The GPIOx_ODR bits can always be accessed directly. The GPIOx_BSRR register provides a way of performing atomic bitwise handling.

There is no need for the software to disable interrupts when programming the GPIOx_ODR at bit level: it is possible to modify one or more bits in a single atomic AHB write access.

### 8.3.6 GPIO locking mechanism

It is possible to freeze the port A and B GPIO control registers by applying a specific write sequence to the GPIOx_LCKR register. The frozen registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.

To write the GPIOx_LCKR register, a specific write / read sequence has to be applied. When the right LOCK sequence is applied to bit 16 in this register, the value of LCKR[15:0] is used to lock the configuration of the I/Os (during the write sequence the LCKR[15:0] value must be the same). When the LOCK sequence has been applied to a port bit, the value of the port bit can no longer be modified until the next MCU reset or peripheral reset. Each GPIOx_LCKR bit freezes the corresponding bit in the control registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.

The LOCK sequence (refer to Section 8.4.8: GPIO port configuration lock register (GPIOx_LCKR) (x = A to B)) can only be performed using a word (32-bit long) access to the GPIOx_LCKR register due to the fact that GPIOx_LCKR bit 16 has to be set at the same time as the [15:0] bits.

For more details refer to LCKR register description in Section 8.4.8: GPIO port configuration lock register (GPIOx_LCKR) (x = A to B).

### 8.3.7 I/O alternate function input/output

Two registers are provided to select one of the alternate function inputs/outputs available for each I/O. With these registers, the user can connect an alternate function to some other pin as required by the application.

This means that a number of possible peripheral functions are multiplexed on each GPIO using the GPIOx_AFRL and GPIOx_AFRH alternate function registers. The application can thus select any one of the possible functions for each I/O. The AF selection signal being common to the alternate function input and alternate function output, a single channel is selected for the alternate function input/output of a given I/O.

For code example refer to Section A.4.2: Alternate function selection sequence on page 728.

To know which functions are multiplexed on each GPIO pin refer to the device datasheet.

### 8.3.8 External interrupt/wake-up lines

All ports have external interrupt capability. To use external interrupt lines, the given pin must not be configured in analog mode or being used as oscillator pin, so the input trigger is kept enabled.
Refer to Section 11.2: Extended interrupts and events controller (EXIT) and to Section 11.2.3: Event management.

8.3.9 Input configuration

When the I/O port is programmed as input:

- The output buffer is disabled
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register provides the I/O state

Figure 15 shows the input configuration of the I/O port bit.

8.3.10 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 whereas 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 whereas a “1” in the output register activates the P-MOS
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value
Figure 16 shows the output configuration of the I/O port bit.

**8.3.11 Alternate function configuration**

When the I/O port is programmed as alternate function:

- The output buffer can be configured in open-drain or push-pull mode
- The output buffer is driven by the signals coming from the peripheral (transmitter enable and data)
- The Schmitt trigger input is activated
- The weak pull-up and pull-down resistors are activated or not depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
Figure 17 shows the alternate function configuration of the I/O port bit.

**Figure 17. Alternate function configuration**

8.3.12 Analog configuration

When the I/O port is programmed as analog configuration:

- The output buffer is disabled
- The Schmitt trigger input is deactivated, 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 by hardware
- Read access to the input data register gets the value “0”

For code example refer to Section A.4.3: Analog GPIO configuration on page 729.
Figure 18 shows the high-impedance, analog-input configuration of the I/O port bits.

**Figure 18. High impedance-analog configuration**

8.3.13 Using the HSE or LSE oscillator pins as GPIOs

When the HSE or LSE oscillator is switched OFF (default state after reset), the related oscillator pins can be used as normal GPIOs.

When the HSE or LSE oscillator is switched ON (by setting the HSEON or LSEON bit in the RCC_CSR register) the oscillator takes control of its associated pins and the GPIO configuration of these pins has no effect.

When the oscillator is configured in a user external clock mode, only the pin is reserved for clock input and the OSC_OUT or OSC32_OUT pin can still be used as normal GPIO.

8.3.14 Using the GPIO pins in the RTC supply domain

The PC13/PC14/PC15 GPIO functionality is lost when the core supply domain is powered off (when the device enters Standby mode). In this case, if their GPIO configuration is not bypassed by the RTC configuration, these pins are set in an analog input mode.

For details about I/O control by the RTC, refer to Section 21.4: RTC functional description.
8.4 GPIO registers

For a summary of register bits, register address offsets and reset values, refer to Table 23. The peripheral registers can be written in word, half word or byte mode.

8.4.1 GPIO port mode register (GPIOx_MODER)  
(x = A to D, F)

Address offset: 0x00
Reset value: 0x2800 0000 for port A
Reset value: 0x0000 0000 for other ports

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


Bits 31:0 MODER[15:0][1:0]: Port x configuration I/O pin y (y = 15 to 0)  
These bits are written by software to configure the I/O mode.  
00: Input mode (reset state)  
01: General purpose output mode  
10: Alternate function mode  
11: Analog mode

8.4.2 GPIO port output type register (GPIOx_OTYPER)  
(x = A to D, F)

Address offset: 0x04
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

OT15 OT14 OT13 OT12 OT11 OT10 OT9 OT8 OT7 OT6 OT5 OT4 OT3 OT2 OT1 OT0

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 OT[15:0]: Port x configuration I/O pin y (y = 15 to 0)  
These bits are written by software to configure the I/O output type.  
0: Output push-pull (reset state)  
1: Output open-drain
8.4.3  GPIO port output speed register (GPIOx_OSPEEDR)
(x = A to D, F)

Address offset: 0x08
Reset value: 0x0C00 0000 (for port A)
Reset value: 0x0000 0000 (for other ports)

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>Bit 30</th>
<th>Bit 29</th>
<th>Bit 28</th>
<th>Bit 27</th>
<th>Bit 26</th>
<th>Bit 25</th>
<th>Bit 24</th>
<th>Bit 23</th>
<th>Bit 22</th>
<th>Bit 21</th>
<th>Bit 20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bit 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSPEEDR15</td>
<td>OSPEEDR14</td>
<td>OSPEEDR13</td>
<td>OSPEEDR12</td>
<td>OSPEEDR11</td>
<td>OSPEEDR10</td>
<td>OSPEEDR9</td>
<td>OSPEEDR8</td>
<td>OSPEEDR7</td>
<td>OSPEEDR6</td>
<td>OSPEEDR5</td>
<td>OSPEEDR4</td>
<td>OSPEEDR3</td>
<td>OSPEEDR2</td>
<td>OSPEEDR1</td>
<td>OSPEEDR0</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:0  **OSPEEDR[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O output speed.
- x0: Low speed
- 01: Medium speed
- 11: High speed

*Note: Refer to the device datasheet for the frequency specifications and the power supply and load conditions for each speed.*

8.4.4  GPIO port pull-up/pull-down register (GPIOx_PUPDR)
(x = A to D, F)

Address offset: 0x0C
Reset value: 0x2400 0000 (for port A)
Reset value: 0x0000 0000 (for other ports)

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>Bit 30</th>
<th>Bit 29</th>
<th>Bit 28</th>
<th>Bit 27</th>
<th>Bit 26</th>
<th>Bit 25</th>
<th>Bit 24</th>
<th>Bit 23</th>
<th>Bit 22</th>
<th>Bit 21</th>
<th>Bit 20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bit 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUPDR15</td>
<td>PUPDR14</td>
<td>PUPDR13</td>
<td>PUPDR12</td>
<td>PUPDR11</td>
<td>PUPDR10</td>
<td>PUPDR9</td>
<td>PUPDR8</td>
<td>PUPDR7</td>
<td>PUPDR6</td>
<td>PUPDR5</td>
<td>PUPDR4</td>
<td>PUPDR3</td>
<td>PUPDR2</td>
<td>PUPDR1</td>
<td>PUPDR0</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:0  **PUPDR[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O pull-up or pull-down
- 00: No pull-up, pull-down
- 01: Pull-up
- 10: Pull-down
- 11: Reserved
8.4.5 GPIO port input data register (GPIOx_IDR)  
(x = A to D, F)  
Address offset: 0x10  
Reset value: 0x0000 XXXX

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>
| 15-0  | IDR[y] (y = 15 to 0): Port x input data I/O pin (y = 15 to 0)  
These bits are read-only. They contain the input value of the corresponding I/O port. |

8.4.6 GPIO port output data register (GPIOx_ODR)  
(x = A to D, F)  
Address offset: 0x14  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>
| 15-0  | ODR[y] (y = 15 to 0): Port output data I/O pin (y = 15 to 0)  
These bits can be read and written by software.  
Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR register (x = A..D, F). |
8.4.7 GPIO port bit set/reset register (GPIOx_BSRR)  
(x = A to D, F)

Address offset: 0x18
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>BR15</th>
<th>BR14</th>
<th>BR13</th>
<th>BR12</th>
<th>BR11</th>
<th>BR10</th>
<th>BR9</th>
<th>BR8</th>
<th>BR7</th>
<th>BR6</th>
<th>BR5</th>
<th>BR4</th>
<th>BR3</th>
<th>BR2</th>
<th>BR1</th>
<th>BR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:16 **BR[15:0]**: Port x reset I/O pin y (y = 15 to 0)  
These bits are write-only. A read to these bits returns the value 0x0000.  
0: No action on the corresponding ODRx bit  
1: Resets the corresponding ODRx bit  

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

Bits 15:0 **BS[15:0]**: Port x set I/O pin y (y = 15 to 0)  
These bits are write-only. A read to these bits returns the value 0x0000.  
0: No action on the corresponding ODRx bit  
1: Sets the corresponding ODRx bit

8.4.8 GPIO port configuration lock register (GPIOx_LCKR)  
(x = A to B)

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, the value of this port bit can no longer be modified until the next MCU reset or peripheral reset.

*Note: A specific write sequence is used to write to the GPIOx_LCKR register. Only word access (32-bit long) is allowed during this locking sequence.  
Each lock bit freezes a specific configuration register (control and alternate function registers).  
Address offset: 0x1C  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>LCK15</th>
<th>LCK14</th>
<th>LCK13</th>
<th>LCK12</th>
<th>LCK11</th>
<th>LCK10</th>
<th>LCK9</th>
<th>LCK8</th>
<th>LCK7</th>
<th>LCK6</th>
<th>LCK5</th>
<th>LCK4</th>
<th>LCK3</th>
<th>LCK2</th>
<th>LCK1</th>
<th>LCK0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
8.4.9 GPIO alternate function low register (GPIOx_AFRL) 
(x = A to D, )

Address offset: 0x20

Reset value: 0x0000 0000

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **LCKK**: Lock key

This bit can be read any time. It can only be modified using the lock key write sequence.

0: Port configuration lock key not active

1: Port configuration lock key active. The GPIOx_LCKR register is locked until the next MCU reset or peripheral reset.

LOCK key write sequence:

WR LCKR[16] = 1 + LCKR[15:0]
WR LCKR[16] = 0 + LCKR[15:0]
WR LCKR[16] = 1 + LCKR[15:0]

RD LCKR

RD LCKR[16] = 1 (this read operation is optional but it confirms that the lock is active)

*Note*: During the LOCK key write sequence, the value of LCK[15:0] must not change.

Any error in the lock sequence aborts the lock.

After the first lock sequence on any bit of the port, any read access on the LCKK bit returns 1 until the next MCU reset or peripheral reset. For code example refer to Section A.4.1: Lock sequence on page 728.

Bits 15:0 **LCK[15:0]**: Port x lock I/O pin y (y = 15 to 0)

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

ABSELy selection:

- 0000: AF0
- 0001: AF1
- 0010: AF2
- 0011: AF3
- 0100: AF4
- 0101: AF5
- 0110: AF6
- 0111: AF7
- 1000: Reserved
- 1001: Reserved
- 1010: Reserved
- 1011: Reserved
- 1100: Reserved
- 1101: Reserved
- 1110: Reserved
- 1111: Reserved
### 8.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A to D, F)

**Address offset:** 0x24  
**Reset value:** 0x0000 0000

#### Bits 31:0 **AFSELy[3:0]:** Alternate function selection for port x pin y (y = 8..15)  
These bits are written by software to configure alternate function I/Os

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>BR15</td>
<td>BR14</td>
<td>BR13</td>
<td>BR12</td>
<td>BR11</td>
<td>BR10</td>
<td>BR9</td>
<td>BR8</td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>

**AFSELy selection:**
- 0000: AF0
- 0001: AF1
- 0010: AF2
- 0011: AF3
- 0100: AF4
- 0101: AF5
- 0110: AF6
- 0111: AF7
- 1000: Reserved
- 1001: Reserved
- 1010: Reserved
- 1011: Reserved
- 1100: Reserved
- 1101: Reserved
- 1110: Reserved
- 1111: Reserved

### 8.4.11 GPIO port bit reset register (GPIOx_BRR) (x = A to D, F)

**Address offset:** 0x28  
**Reset value:** 0x0000 0000

#### Bits 31:16 Reserved, must be kept at reset value.

#### Bits 15:0 **BR[y][15:0]:** Port x reset IO pin y (y = 15 to 0)  
These bits are write-only. A read to these bits returns the value 0x0000.  
0: No action on the corresponding ODx bit  
1: Reset the corresponding ODx bit
## 8.4.12 GPIO register map

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

| Offset | Register name             | 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   | GPIOA_MODER              | 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   |
|        | (where x = B..D, F)     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | GPIOx_OTYPER             |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x04   | (where x = A..D, F)     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | GPIOA_OSPEEDR            |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x08   | (where x = B..D, F)     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | GPIOx_OSPEEDR            |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x0C   | GPIOA_PUPDR              |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | (where x = B..D, F)     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x10   | GPIOx_IDR                |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x14   | (where x = A..D, F)     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x18   | GPIOx_BSR                 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

140/775   RM0360 Rev 5
Refer to Section 2.2 on page 37 for the register boundary addresses.
9 System configuration controller (SYSCFG)

The devices feature a set of configuration registers. The main purposes of the system configuration controller are the following:

- Enabling/disabling I2C Fast Mode Plus on some IO ports
- Remapping some DMA trigger sources to different DMA channels
- Remapping the memory located at the beginning of the code area
- Managing the external interrupt line connection to the GPIOs
- Managing robustness feature

9.1 SYSCFG registers

9.1.1 SYSCFG configuration register 1 (SYSCFG_CFGR1)

This register is used for specific configurations of memory and DMA requests remap and to control special I/O features.

Two bits are used to configure the type of memory accessible at address 0x0000 0000. These bits are used to select the physical remap by software and so, bypass the hardware BOOT selection.

After reset these bits take the value selected by the actual boot mode configuration.

Address offset: 0x00
Reset value: 0x0000 000X (X is the memory mode selected by the actual boot mode configuration

Note: For STM32F030xC devices, DMA remapping bits are replaced by more flexible mapping configured through DMA_CSELR register. Refer to Section 10.6.7: DMA channel selection register (DMA_CSELR) for more details.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Res.</td>
<td>Res.</td>
<td>Res.</td>
<td>Res.</td>
<td>TIM17_DMA_RMP</td>
<td>TIM16_DMA_RMP</td>
<td>USART1_RX_DMA_RMP</td>
<td>USART1_TX_DMA_RMP</td>
<td>ADC_DMA_RMP</td>
<td>Res.</td>
<td>Res.</td>
<td>PA11_PA12_RMP</td>
<td>MEM_MODE [1:0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:27  Reserved, must be kept at reset value.

Bit 26 **USART3_DMA_RMP**: USART3 DMA request remapping bit. Available on STM32F070xB devices only.

This bit is set and cleared by software. It controls the remapping of USART3 DMA requests.

0: Disabled, need to enable remap before use.

1: Remap (USART3_RX and USART3_TX DMA requests mapped on DMA channel 3 and 2 respectively)
**Bits 25:24**  Reserved, must be kept at reset value.

**Bits 23:22**  **I2C_PAx_FMP:** Fast Mode Plus (FM+) driving capability activation bits. Available on STM32F030x4, STM32F030x6, STM32F070x6 and STM32F030xC devices only. These bits are set and cleared by software. Each bit enables I2C FM+ mode for PA10 and PA9 I/Os.
   - 0: PAx pin operates in standard mode.
   - 1: I2C FM+ mode enabled on PAx pin and the Speed control is bypassed.

**Bit 21**  **I2C2_FMP:** FM+ driving capability activation for I2C2. Available on STM32F070xB and STM32F030xC devices only.
   - This bit is set and cleared by software. This bit is OR-ed with I2C_Pxx_FM+ bits.
   - 0: FM+ mode is controlled by I2C_Pxx_FM+ bits only.
   - 1: FM+ mode is enabled on all I2C2 pins selected through selection bits in GPIOx_AFR registers. This is the only way to enable the FM+ mode for pads without a dedicated I2C_Pxx_FM+ control bit.

**Bit 20**  **I2C1_FMP:** FM+ driving capability activation for I2C1. Not available on STM32F030x8 devices.
   - This bit is set and cleared by software. This bit is OR-ed with I2C_Pxx_FM+ bits.
   - 0: FM+ mode is controlled by I2C_Pxx_FM+ bits only.
   - 1: FM+ mode is enabled on all I2C1 pins selected through selection bits in GPIOx_AFR registers. This is the only way to enable the FM+ mode for pads without a dedicated I2C_Pxx_FM+ control bit.

**Bits 19:16**  **I2C_PBx_FMP:** Fast Mode Plus (FM+) driving capability activation bits.
   - These bits are set and cleared by software. Each bit enables I2C FM+ mode for PB6, PB7, PB8, and PB9 I/Os.
   - 0: PBx pin operates in standard mode.
   - 1: I2C FM+ mode enabled on PBx pin and the Speed control is bypassed.

**Bits 15:13**  Reserved, must be kept at reset value.

**Bit 12**  **TIM17_DMA_RMP:** TIM17 DMA request remapping bit. Available on STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices only.
   - This bit is set and cleared by software. It controls the remapping of TIM17 DMA requests.
   - 0: No remap (TIM17_CH1 and TIM17_UP DMA requests mapped on DMA channel 1)
   - 1: Remap (TIM17_CH1 and TIM17_UP DMA requests mapped on DMA channel 2)

**Bit 11**  **TIM16_DMA_RMP:** TIM16 DMA request remapping bit. Available on STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices only.
   - This bit is set and cleared by software. It controls the remapping of TIM16 DMA requests.
   - 0: No remap (TIM16_CH1 and TIM16_UP DMA requests mapped on DMA channel 3)
   - 1: Remap (TIM16_CH1 and TIM16_UP DMA requests mapped on DMA channel 4)

**Bit 10**  **USART1_RX_DMA_RMP:** USART1_RX DMA request remapping bit. Available on STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices only.
   - This bit is set and cleared by software. It controls the remapping of USART1_RX DMA requests.
   - 0: No remap (USART1_RX DMA request mapped on DMA channel 3)
   - 1: Remap (USART1_RX DMA request mapped on DMA channel 5)
9.1.2 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>Bit 30</th>
<th>Bit 29</th>
<th>Bit 28</th>
<th>Bit 27</th>
<th>Bit 26</th>
<th>Bit 25</th>
<th>Bit 24</th>
<th>Bit 23</th>
<th>Bit 22</th>
<th>Bit 21</th>
<th>Bit 20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bit 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bit 9 **USART1_TX_DMA_RMP**: USART1_TX DMA request remapping bit. Available on STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices only. This bit is set and cleared by software. It controls the remapping of USART1_TX DMA requests.

0: No remap (USART1_TX DMA request mapped on DMA channel 2)
1: Remap (USART1_TX DMA request mapped on DMA channel 4)

Bit 8 **ADC_DMA_RMP**: ADC DMA request remapping bit. Available on STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices only. This bit is set and cleared by software. It controls the remapping of ADC DMA requests.

0: No remap (ADC DMA request mapped on DMA channel 1)
1: Remap (ADC DMA request mapped on DMA channel 2)

Bits 7:5 Reserved, must be kept at reset value.

Bit 4 **PA11_PA12_RMP**: PA11 and PA12 remapping bit for small packages (28 and 20 pins). Available on STM32F070x6 devices only. This bit is set and cleared by software. It controls the remapping of either PA9/10 or PA11/12 pin pair on small pin-count packages.

0: No remap (pin pair PA9/10 mapped on the pins)
1: Remap (pin pair PA11/12 mapped instead of PA9/10)

Bits 3:2 Reserved, must be kept at reset value.

Bits 1:0 **MEM_MODE[1:0]**: Memory mapping selection bits

These bits are set and cleared by software. They control the memory internal mapping at address 0x0000 0000. After reset these bits take on the value selected by the actual boot mode configuration. Refer to Section 2.5: Boot configuration for more details.

x0: Main Flash memory mapped at 0x0000 0000
01: System Flash memory mapped at 0x0000 0000
11: Embedded SRAM mapped at 0x0000 0000

STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8 and STM32F070xB devices only.
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration bits (x = 0 to 3)
These bits are written by software to select the source input for the EXTIx external interrupt.
- x000: PA[x] pin
- x001: PB[x] pin
- x010: PC[x] pin
- x011: PD[x] pin
- x100: Reserved
- x101: PF[x] pin
- other configurations: reserved

**Note:** Some of the I/O pins mentioned in the above register may not be available on small packages.

### 9.1.3 SYSCFG external interrupt configuration register 2
**(SYSCFG_EXTICR2)**

Address offset: 0x0C
Reset value: 0x0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**EXTI7[3:0]**  **EXTI6[3:0]**  **EXTI5[3:0]**  **EXTI4[3:0]**

erw  erw  erw  erw  erw  erw  erw  erw  erw  erw  erw  erw  erw  erw  erw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration bits (x = 4 to 7)
These bits are written by software to select the source input for the EXTIx external interrupt.
- x000: PA[x] pin
- x001: PB[x] pin
- x010: PC[x] pin
- x011: PD[x] pin
- x100: Reserved
- x101: PF[x] pin
- other configurations: reserved

**Note:** Some of the I/O pins mentioned in the above register may not be available on small packages.

### 9.1.4 SYSCFG external interrupt configuration register 3
**(SYSCFG_EXTICR3)**

Address offset: 0x10
Reset value: 0x0000
## System configuration controller (SYSCFG)

### 9.1.5 SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

Address offset: 0x14

Reset value: 0x0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration bits (x = 8 to 11)

These bits are written by software to select the source input for the EXTIx external interrupt.

- x000: PA[x] pin
- x001: PB[x] pin
- x010: PC[x] pin
- x011: PD[x] pin
- x100: Reserved
- x101: PF[x] pin
- other configurations: reserved

### Note:
Some of the I/O pins mentioned in the above register may not be available on small packages.

---

146/775 RM0360 Rev 5
## 9.1.6 SYSCFG configuration register 2 (SYSCFG_CFGR2)

Address offset: 0x18

System reset value: 0x0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rc_w1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Bits 31:9 Reserved, must be kept at reset value

#### Bit 8 SRAM_PEF: SRAM parity error flag
This bit is set by hardware when an SRAM parity error is detected. It is cleared by software by writing '1'.
- 0: No SRAM parity error detected
- 1: SRAM parity error detected

#### Bits 7:2 Reserved, must be kept at reset value

#### Bit 1 SRAM_PARITY_LOCK: SRAM parity lock bit
This bit is set by software and cleared by a system reset. It can be used to enable and lock the SRAM parity error signal connection to TIM1/15/16/17 Break input.
- 0: SRAM parity error disconnected from TIM1/15/16/17 Break input
- 1: SRAM parity error connected to TIM1/15/16/17 Break input

#### Bit 0 LOCKUP_LOCK: Cortex-M0 LOCKUP bit enable bit
This bit is set by software and cleared by a system reset. It can be used to enable and lock the connection of Cortex-M0 LOCKUP (Hardfault) output to TIM1/15/16/17 Break input.
- 0: Cortex-M0 LOCKUP output disconnected from TIM1/15/16/17 Break input
- 1: Cortex-M0 LOCKUP output connected to TIM1/15/16/17 Break input
9.1.7 SYSCFG register maps

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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>SYSCFG_CFGR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x08</td>
<td>SYSCFG_EXTICR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x0C</td>
<td>SYSCFG_EXTICR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x10</td>
<td>SYSCFG_EXTICR3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x14</td>
<td>SYSCFG_EXTICR4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x18</td>
<td>SYSCFG_CFGR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Refer to Section 2.2 on page 37 for the register boundary addresses.
10 Direct memory access controller (DMA)

10.1 Introduction

The direct memory access (DMA) controller is a bus master and system peripheral. The DMA is used to perform programmable data transfers between memory-mapped peripherals and/or memories, upon the control of an off-loaded CPU. The DMA controller features a single AHB master architecture. There is one instance of DMA with 5 channels. Each channel is dedicated to managing memory access requests from one or more peripherals. The DMA includes an arbiter for handling the priority between DMA requests.

10.2 DMA main features

- Single AHB master
- Peripheral-to-memory, memory-to-peripheral, memory-to-memory and peripheral-to-peripheral data transfers
- Access, as source and destination, to on-chip memory-mapped devices such as flash memory, SRAM, and AHB and APB peripherals
- All DMA channels independently configurable:
  - Each channel is associated either with a DMA request signal coming from a peripheral, or with a software trigger in memory-to-memory transfers. This configuration is done by software.
  - Priority between the requests is programmable by software (4 levels per channel: very high, high, medium, low) and by hardware in case of equality (such as request to channel 1 has priority over request to channel 2).
  - Transfer size of source and destination are independent (byte, half-word, word), emulating packing and unpacking. Source and destination addresses must be aligned on the data size.
  - Support of transfers from/to peripherals to/from memory with circular buffer management
  - Programmable number of data to be transferred: 0 to $2^{16} - 1$
- Generation of an interrupt request per channel. Each interrupt request is caused from any of the three DMA events: transfer complete, half transfer, or transfer error.
10.3 DMA implementation

10.3.1 DMA

DMA is implemented with the hardware configuration parameters shown in the table below.

<table>
<thead>
<tr>
<th>Feature</th>
<th>DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of channels</td>
<td>5</td>
</tr>
</tbody>
</table>

10.3.2 DMA request mapping

DMA controller

The hardware requests from the peripherals are simply logically ORed before entering the DMA. This means that on one channel, only one request must be enabled at a time (see Figure 19).

The peripheral DMA requests can be independently activated/de-activated by programming the DMA control bit in the registers of the corresponding peripheral.

**Caution:** A same peripheral request can be assigned to two different channels only if the application ensures that these channels are not requested to be served at the same time. In other words, if two different channels receive a same asserted peripheral request at the same time, an unpredictable DMA hardware behavior occurs.

*Table 26* and *Table 27* list the DMA requests for each channel and alternate position.
1. DMA request mapped on this DMA channel only if the corresponding remapping bit is cleared in SYSCFG configuration register 1 (SYSCFG_CFGR1).

2. DMA request mapped on this DMA channel only if the corresponding remapping bit is set in SYSCFG configuration register 1 (SYSCFG_CFGR1).

Table 26. DMA requests for each channel on STM32F030x4/x6/x8 and STM32F070x6/xB devices

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Channel 1</th>
<th>Channel 2</th>
<th>Channel 3</th>
<th>Channel 4</th>
<th>Channel 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>ADC(1)</td>
<td>ADC(2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SPI</td>
<td>-</td>
<td>SPI1_RX</td>
<td>SPI1_TX</td>
<td>SPI2_RX</td>
<td>SPI2_TX</td>
</tr>
<tr>
<td>USART</td>
<td>-</td>
<td>USART1_TX(1)</td>
<td>USAART1_RX(1)</td>
<td>USAART1_RX(2)</td>
<td>USAART1_RX(2)</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>USART3_TX(2)</td>
<td>USAART3_RX(2)</td>
<td>USAART2_TX</td>
<td>USAART2_RX</td>
</tr>
<tr>
<td>I2C</td>
<td>-</td>
<td>I2C1_TX</td>
<td>I2C1_RX</td>
<td>I2C2_TX</td>
<td>I2C2_RX</td>
</tr>
</tbody>
</table>
Table 26. DMA requests for each channel on STM32F030x4/x6/x8 and STM32F070x6/xB devices (continued)

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Channel 1</th>
<th>Channel 2</th>
<th>Channel 3</th>
<th>Channel 4</th>
<th>Channel 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM1</td>
<td>-</td>
<td>TIM1_CH1</td>
<td>TIM1_CH2</td>
<td>TIM1_CH4</td>
<td>TIM1_CH3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TIM1_TRIG</td>
<td>TIM1_UP</td>
</tr>
<tr>
<td>TIM3</td>
<td>-</td>
<td>TIM3_CH3</td>
<td>TIM3_CH4</td>
<td>TIM3_CH1</td>
<td>-</td>
</tr>
<tr>
<td>TIM6</td>
<td>-</td>
<td>-</td>
<td>TIM6_UP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TIM7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>TIM7_UP</td>
<td>-</td>
</tr>
<tr>
<td>TIM15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>TIM15_CH1</td>
</tr>
<tr>
<td>TIM16</td>
<td>TIM16_CH1</td>
<td>TIM16_UP</td>
<td>TIM16_CH1</td>
<td>TIM16_UP</td>
<td>-</td>
</tr>
<tr>
<td>TIM17</td>
<td>TIM17_CH1</td>
<td>TIM17_UP</td>
<td>TIM17_CH1</td>
<td>TIM17_UP</td>
<td>-</td>
</tr>
</tbody>
</table>

1. DMA request mapped on this DMA channel only if the corresponding remapping bit is cleared in SYSCFG configuration register 1 (SYSCFG_CFGR1).
2. DMA request mapped on this DMA channel only if the corresponding remapping bit is set in SYSCFG configuration register 1 (SYSCFG_CFGR1).

Table 27. DMA requests for each channel on STM32F030xC devices

<table>
<thead>
<tr>
<th>CxS[3:0]</th>
<th>Channel 1</th>
<th>Channel 2</th>
<th>Channel 3</th>
<th>Channel 4</th>
<th>Channel 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>-</td>
<td>TIM3_CH3</td>
<td>TIM3_CH4</td>
<td>TIM1_CH4</td>
<td>TIM1_UP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TIM3_UP</td>
<td>TIM1_TRIG</td>
<td>TIM1_COM</td>
</tr>
<tr>
<td>ADC</td>
<td>-</td>
<td>-</td>
<td>TIM6_UP</td>
<td>TIM7_UP</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>USART1_TX</td>
<td>USART1_RX</td>
<td>USART2_TX</td>
<td>USART2_RX</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>SPI1_RX</td>
<td>SPI1_TX</td>
<td>SPI2_RX</td>
<td>SPI2_TX</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>I2C1_TX</td>
<td>I2C1_RX</td>
<td>I2C2_TX</td>
<td>I2C2_RX</td>
</tr>
<tr>
<td></td>
<td>TIM1_CH1</td>
<td>TIM1_CH2</td>
<td>-</td>
<td>-</td>
<td>TIM1_CH3</td>
</tr>
<tr>
<td>TIM17_CH1</td>
<td>TIM17_UP</td>
<td>-</td>
<td>TIM16_CH1</td>
<td>TIM16_UP</td>
<td>TIM3_CH1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TIM17_UP</td>
<td>TIM16_TRIG</td>
<td>-</td>
</tr>
<tr>
<td>0001</td>
<td>ADC</td>
<td>ADC</td>
<td>TIM6_UP</td>
<td>TIM7_UP</td>
<td>-</td>
</tr>
<tr>
<td>0010</td>
<td>-</td>
<td>I2C1_TX</td>
<td>I2C1_RX</td>
<td>I2C2_TX</td>
<td>I2C2_RX</td>
</tr>
<tr>
<td>0011</td>
<td>-</td>
<td>SPI1_RX</td>
<td>SPI1_TX</td>
<td>SPI2_RX</td>
<td>SPI2_TX</td>
</tr>
<tr>
<td>0100</td>
<td>-</td>
<td>TIM1_CH1</td>
<td>TIM1_CH2</td>
<td>-</td>
<td>TIM1_CH3</td>
</tr>
</tbody>
</table>
10.4 DMA functional description

10.4.1 DMA block diagram

The DMA block diagram is shown in the figure below.

![DMA block diagram](image)

The DMA controller performs direct memory transfer by sharing the AHB system bus with other system masters. The bus matrix implements round-robin scheduling. DMA requests...
may stop the CPU access to the system bus for a number of bus cycles, when CPU and 
DMA target the same destination (memory or peripheral).

According to its configuration through the AHB slave interface, the DMA controller arbitrates 
between the DMA channels and their associated received requests. The DMA controller 
also schedules the DMA data transfers over the single AHB port master.

The DMA controller generates an interrupt per channel to the interrupt controller.

10.4.2 DMA transfers

The software configures the DMA controller at channel level, in order to perform a block 
transfer, composed of a sequence of AHB bus transfers.

A DMA block transfer may be requested from a peripheral, or triggered by the software in 
case of memory-to-memory transfer.

After an event, the following steps of a single DMA transfer occur:

1. The peripheral sends a single DMA request signal to the DMA controller.
2. The DMA controller serves the request, depending on the priority of the channel 
   associated to this peripheral request.
3. As soon as the DMA controller grants the peripheral, an acknowledge is sent to the 
   peripheral by the DMA controller.
4. The peripheral releases its request as soon as it gets the acknowledge from the DMA 
   controller.
5. Once the request is de-asserted by the peripheral, the DMA controller releases the 
   acknowledge.

The peripheral may order a further single request and initiate another single DMA transfer.

The request/acknowledge protocol is used when a peripheral is either the source or the 
destination of the transfer. For example, in case of memory-to-peripheral transfer, the 
peripheral initiates the transfer by driving its single request signal to the DMA controller. The 
DMA controller reads then a single data in the memory and writes this data to the peripheral.

For a given channel x, a DMA block transfer consists of a repeated sequence of:

- a single DMA transfer, encapsulating two AHB transfers of a single data, over the DMA 
  AHB bus master:
  - a single data read (byte, half-word or word) from the peripheral data register or a 
    location in the memory, addressed through an internal current peripheral/memory 
    address register.
    The start address used for the first single transfer is the base address of the 
    peripheral or memory, and is programmed in the DMA_CPARx or DMA_CMARx 
    register.
  - a single data write (byte, half-word or word) to the peripheral data register or a 
    location in the memory, addressed through an internal current peripheral/memory 
    address register.
    The start address used for the first transfer is the base address of the peripheral or 
    memory, and is programmed in the DMA_CPARx or DMA_CMARx register.

- post-decrementing of the programmed DMA_CNDTRx register 
  This register contains the remaining number of data items to transfer (number of AHB 'read followed by write' transfers).

This sequence is repeated until DMA_CNDTRx is null.
**Note:** The AHB master bus source/destination address must be aligned with the programmed size of the transferred single data to the source/destination.

### 10.4.3 DMA arbitration

The DMA arbiter manages the priority between the different channels.

When an active channel \( x \) is granted by the arbiter (hardware requested or software triggered), a single DMA transfer is issued (such as an AHB ‘read followed by write’ transfer of a single data). Then, the arbiter considers again the set of active channels and selects the one with the highest priority.

The priorities are managed in two stages:

- **software**: priority of each channel is configured in the DMA\_CCR\(x\) register, to one of the four different levels:
  - very high
  - high
  - medium
  - low

- **hardware**: if two requests have the same software priority level, the channel with the lowest index gets priority. For example, channel 2 gets priority over channel 4.

When a channel \( x \) is programmed for a block transfer in memory-to-memory mode, arbitration is considered between each single DMA transfer of this channel \( x \). Whenever there is another concurrent active requested channel, the DMA arbiter automatically alternates and grants the other highest-priority requested channel, which may be of lower priority than the memory-to-memory channel.

### 10.4.4 DMA channels

Each channel may handle a DMA transfer between a peripheral register located at a fixed address, and a memory address. The amount of data items to transfer is programmable. The register that contains the amount of data items to transfer is decremented after each transfer.

A DMA channel is programmed at block transfer level.

**Programmable data sizes**

The transfer sizes of a single data (byte, half-word, or word) to the peripheral and memory are programmable through, respectively, the PSIZE[1:0] and MSIZE[1:0] fields of the DMA\_CCR\(x\) register.

**Pointer incrementation**

The peripheral and memory pointers may be automatically incremented after each transfer, depending on the PINC and MINC bits of the DMA\_CCR\(x\) register.

If the **incremented mode** is enabled (PINC or MINC set to 1), the address of the next transfer is the address of the previous one incremented by 1, 2 or 4, depending on the data size defined in PSIZE[1:0] or MSIZE[1:0]. The first transfer address is the one programmed in the DMA\_CPAR\(x\) or DMA\_CMAR\(x\) register. During transfers, these registers keep the initially programmed value. The current transfer addresses (in the current internal peripheral/memory address register) are not accessible by software.
If the channel x is configured in **non-circular mode**, no DMA request is served after the last data transfer (once the number of single data to transfer reaches zero). The DMA channel must be disabled in order to reload a new number of data items into the DMA_CNDTRx register.

**Note:** *If the channel x is disabled, the DMA registers are not reset. The DMA channel registers (DMA_CCRx, DMA_CPARx and DMA_CMARx) retain the initial values programmed during the channel configuration phase.*

In **circular mode**, after the last data transfer, the DMA_CNDTRx register is automatically reloaded with the initially programmed value. The current internal address registers are reloaded with the base address values from the DMA_CPARx and DMA_CMARx registers.

**Channel configuration procedure**

The following sequence is needed to configure a DMA channel x:

1. Set the peripheral register address in the DMA_CPARx register.  
   The data is moved from/to this address to/from the memory after the peripheral event, or after the channel is enabled in memory-to-memory mode.

2. Set the memory address in the DMA_CMARx register.  
   The data is written to/read from the memory after the peripheral event or after the channel is enabled in memory-to-memory mode.

3. Configure the total number of data to transfer in the DMA_CNDTRx register.  
   After each data transfer, this value is decremented.

4. Configure the parameters listed below in the DMA_CCRx register:  
   – the channel priority  
   – the data transfer direction  
   – the circular mode  
   – the peripheral and memory incremented mode  
   – the peripheral and memory data size  
   – the interrupt enable at half and/or full transfer and/or transfer error

5. Activate the channel by setting the EN bit in the DMA_CCRx register.

For code example refer to the Appendix *Section A.5.1: DMA channel configuration sequence.*

A channel, as soon as enabled, may serve any DMA request from the peripheral connected to this channel, or may start a memory-to-memory block transfer.

**Note:** *The two last steps of the channel configuration procedure may be merged into a single access to the DMA_CCRx register, to configure and enable the channel.*

**Channel state and disabling a channel**

A channel x in active state is an enabled channel (read DMA_CCRx.EN = 1). An active channel x is a channel that must have been enabled by the software (DMA_CCRx.EN set to 1) and afterwards with no occurred transfer error (DMA_ISR.TEIFx = 0). In case there is a transfer error, the channel is automatically disabled by hardware (DMA_CCRx.EN = 0).
The three following use cases may happen:

- **Suspend and resume a channel**
  This corresponds to the two following actions:
  - An active channel is disabled by software (writing DMA_CCRx.EN = 0 whereas DMA_CCRx.EN = 1).
  - The software enables the channel again (DMA_CCRx.EN set to 1) without reconfiguring the other channel registers (such as DMA_CNDTRx, DMA_CPARx and DMA_CMARx).
  This case is not supported by the DMA hardware, that does not guarantee that the remaining data transfers are performed correctly.

- **Stop and abort a channel**
  If the application does not need any more the channel, this active channel can be disabled by software. The channel is stopped and aborted but the DMA_CNDTRx register content may not correctly reflect the remaining data transfers versus the aborted source and destination buffer/register.

- **Abort and restart a channel**
  This corresponds to the software sequence: disable an active channel, then reconfigure the channel and enable it again.
  This is supported by the hardware if the following conditions are met:
  - The application guarantees that, when the software is disabling the channel, a DMA data transfer is not occurring at the same time over its master port. For example, the application can first disable the peripheral in DMA mode, in order to ensure that there is no pending hardware DMA request from this peripheral.
  - The software must operate separated write accesses to the same DMA_CCRx register: First disable the channel. Second reconfigure the channel for a next block transfer including the DMA_CCRx if a configuration change is needed. There are read-only DMA_CCRx register fields when DMA_CCRx.EN=1. Finally enable again the channel.

When a channel transfer error occurs, the EN bit of the DMA_CCRx register is cleared by hardware. This EN bit can not be set again by software to re-activate the channel x, until the TEIFx bit of the DMA_ISR register is set.

**Circular mode (in memory-to-peripheral/peripheral-to-memory transfers)**

The circular mode is available to handle circular buffers and continuous data flows (such as ADC scan mode). This feature is enabled using the CIRC bit in the DMA_CCRx register.

*Note:* The circular mode must not be used in memory-to-memory mode. Before enabling a channel in circular mode (CIRC = 1), the software must clear the MEM2MEM bit of the DMA_CCRx register. When the circular mode is activated, the amount of data to transfer is automatically reloaded with the initial value programmed during the channel configuration phase, and the DMA requests continue to be served.

In order to stop a circular transfer, the software needs to stop the peripheral from generating DMA requests (such as quit the ADC scan mode), before disabling the DMA channel. The software must explicitly program the DMA_CNDTRx value before starting/enabling a transfer, and after having stopped a circular transfer.
Memory-to-memory mode

The DMA channels may operate without being triggered by a request from a peripheral. This mode is called memory-to-memory mode, and is initiated by software.

If the MEM2MEM bit in the DMA_CCRx register is set, the channel, if enabled, initiates transfers. The transfer stops once the DMA_CNDTRx register reaches zero.

Note: The memory-to-memory mode must not be used in circular mode. Before enabling a channel in memory-to-memory mode (MEM2MEM = 1), the software must clear the CIRC bit of the DMA_CCRx register.

Peripheral-to-peripheral mode

Any DMA channel can operate in peripheral-to-peripheral mode:

- when the hardware request from a peripheral is selected to trigger the DMA channel
  This peripheral is the DMA initiator and paces the data transfer from/to this peripheral to/from a register belonging to another memory-mapped peripheral (this one being not configured in DMA mode).
- when no peripheral request is selected and connected to the DMA channel
  The software configures a register-to-register transfer by setting the MEM2MEM bit of the DMA_CCRx register.

Programming transfer direction, assigning source/destination

The value of the DIR bit of the DMA_CCRx register sets the direction of the transfer, and consequently, it identifies the source and the destination, regardless the source/destination type (peripheral or memory):

- **DIR = 1** defines typically a memory-to-peripheral transfer. More generally, if DIR = 1:
  - The **source** attributes are defined by the DMA_MARx register, the MSIZE[1:0] field and MINC bit of the DMA_CCRx register.
    Regardless of their usual naming, these ‘memory’ register, field and bit are used to define the source peripheral in peripheral-to-peripheral mode.
  - The **destination** attributes are defined by the DMA_PARx register, the PSIZE[1:0] field and PINC bit of the DMA_CCRx register.
    Regardless of their usual naming, these ‘peripheral’ register, field and bit are used to define the destination memory in memory-to-memory mode.

- **DIR = 0** defines typically a peripheral-to-memory transfer. More generally, if DIR = 0:
  - The **source** attributes are defined by the DMA_PARx register, the PSIZE[1:0] field and PINC bit of the DMA_CCRx register.
    Regardless of their usual naming, these ‘peripheral’ register, field and bit are used to define the source memory in memory-to-memory mode.
  - The **destination** attributes are defined by the DMA_MARx register, the MSIZE[1:0] field and MINC bit of the DMA_CCRx register.
    Regardless of their usual naming, these ‘memory’ register, field and bit are used to define the destination peripheral in peripheral-to-peripheral mode.
10.4.5 DMA data width, alignment and endianness

When PSIZE[1:0] and MSIZE[1:0] are not equal, the DMA controller performs some data alignments as described in the table below.

Table 28. Programmable data width and endian behavior (when PINC = MINC = 1)

<table>
<thead>
<tr>
<th>Source port width (MSIZE if DIR = 1, else PSIZE)</th>
<th>Destination port width (PSIZE if DIR = 1, else MSIZE)</th>
<th>Number of data items to transfer (NDT)</th>
<th>Source content: address / data (DMA_CMARx if DIR = 1, else DMA_CPARx)</th>
<th>Destination content: address / data (DMA_CPARx if DIR = 1, else DMA_CMARx)</th>
<th>DMA transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 8 4</td>
<td>8 8 4</td>
<td>@0x0 / B0</td>
<td>1: read B0[7:0] then write B0[7:0] @0x0</td>
<td>@0x0 / B0</td>
<td>1. @0x0 / B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x1 / B1</td>
<td>2: read B1[7:0] then write B1[7:0] @0x1</td>
<td></td>
<td>2. @0x1 / B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x2 / B2</td>
<td>3: read B2[7:0] then write B2[7:0] @0x2</td>
<td></td>
<td>3. @0x2 / B2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x3 / B3</td>
<td>4: read B3[7:0] then write B3[7:0] @0x3</td>
<td></td>
<td>4. @0x3 / B3</td>
</tr>
<tr>
<td>8 16 4</td>
<td>@0x0 / B0</td>
<td>1: read B0[7:0] then write 0000B0[15:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / 00B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x1 / B1</td>
<td>2: read B1[7:0] then write 0000B1[15:0] @0x2</td>
<td></td>
<td>2. @0x2 / 00B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x2 / B2</td>
<td>3: read B2[7:0] then write 0000B2[15:0] @0x4</td>
<td></td>
<td>3. @0x4 / 00B2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x3 / B3</td>
<td>4: read B3[7:0] then write 0000B3[15:0] @0x6</td>
<td></td>
<td>4. @0x6 / 00B3</td>
</tr>
<tr>
<td>8 32 4</td>
<td>@0x0 / B0</td>
<td>1: read B0[7:0] then write 00000000B0[31:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / 000000B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x1 / B1</td>
<td>2: read B1[7:0] then write 00000000B1[31:0] @0x4</td>
<td></td>
<td>2. @0x4 / 000000B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x2 / B2</td>
<td>3: read B2[7:0] then write 00000000B2[31:0] @0x8</td>
<td></td>
<td>3. @0x8 / 000000B2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x3 / B3</td>
<td>4: read B3[7:0] then write 00000000B3[31:0] @0xC</td>
<td></td>
<td>4. @0xC / 000000B3</td>
</tr>
<tr>
<td>16 8 4</td>
<td>@0x0 / B1B0</td>
<td>1: read B1B0[15:0] then write B0[7:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x2 / B3B2</td>
<td>2: read B3B2[15:0] then write B2[7:0] @0x1</td>
<td></td>
<td>2. @0x1 / B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x4 / B5B4</td>
<td>3: read B5B4[15:0] then write B4[7:0] @0x2</td>
<td></td>
<td>3. @0x2 / B4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x6 / B7B6</td>
<td>4: read B7B6[15:0] then write B6[7:0] @0x3</td>
<td></td>
<td>4. @0x3 / B6</td>
</tr>
<tr>
<td>16 16 4</td>
<td>@0x0 / B1B0</td>
<td>1: read B1B0[15:0] then write B1B0[15:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / B1B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x2 / B3B2</td>
<td>2: read B3B2[15:0] then write B3B2[15:0] @0x2</td>
<td></td>
<td>2. @0x2 / B3B2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x4 / B5B4</td>
<td>3: read B5B4[15:0] then write B5B4[15:0] @0x4</td>
<td></td>
<td>3. @0x4 / B5B4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x6 / B7B6</td>
<td>4: read B7B6[15:0] then write B7B6[15:0] @0x6</td>
<td></td>
<td>4. @0x6 / B7B6</td>
</tr>
<tr>
<td>16 32 4</td>
<td>@0x0 / B1B0</td>
<td>1: read B1B0[15:0] then write 00000000B1B0[31:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / 000000B1B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x2 / B3B2</td>
<td>2: read B3B2[15:0] then write 00000000B3B2[31:0] @0x4</td>
<td></td>
<td>2. @0x4 / 000000B3B2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x4 / B5B4</td>
<td>3: read B5B4[15:0] then write 00000000B5B4[31:0] @0x8</td>
<td></td>
<td>3. @0x8 / 000000B5B4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x6 / B7B6</td>
<td>4: read B7B6[15:0] then write 00000000B7B6[31:0] @0xC</td>
<td></td>
<td>4. @0xC / 000000B7B6</td>
</tr>
<tr>
<td>32 8 4</td>
<td>@0x0 / B3B2B1B0</td>
<td>1: read B3B2B1B0[31:0] then write B0[7:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x4 / B7B6B5B4</td>
<td>2: read B7B6B5B4[31:0] then write B4[7:0] @0x1</td>
<td></td>
<td>2. @0x1 / B4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x8 / BBBABAB9B8</td>
<td>3: read BBBABAB9B8[31:0] then write B6[7:0] @0x2</td>
<td></td>
<td>3. @0x2 / B8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0xC / BFBEBCBDC</td>
<td>4: read BFBEBCBDC[31:0] then write BC[7:0] @0x3</td>
<td></td>
<td>4. @0xC / BC</td>
</tr>
<tr>
<td>32 16 4</td>
<td>@0x0 / B3B2B1B0</td>
<td>1: read B3B2B1B0[31:0] then write B1B0[15:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / B1B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x4 / B7B6B5B4</td>
<td>2: read B7B6B5B4[31:0] then write B5B4[15:0] @0x2</td>
<td></td>
<td>2. @0x2 / B5B4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x8 / BBBABAB9B8</td>
<td>3: read BBBABAB9B8[31:0] then write B9B8[15:0] @0x4</td>
<td></td>
<td>3. @0x4 / B9B8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0xC / BFBEBCBDC</td>
<td>4: read BFBEBCBDC[31:0] then write BDBC[15:0] @0x6</td>
<td></td>
<td>4. @0xC / BDBC</td>
</tr>
<tr>
<td>32 32 4</td>
<td>@0x0 / B3B2B1B0</td>
<td>1: read B3B2B1B0[31:0] then write B3B2B1B0[31:0] @0x0</td>
<td></td>
<td></td>
<td>1. @0x0 / B3B2B1B0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x4 / B7B6B5B4</td>
<td>2: read B7B6B5B4[31:0] then write B7B6B5B4[31:0] @0x4</td>
<td></td>
<td>2. @0x4 / B7B6B5B4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0x8 / BBBABAB9B8</td>
<td>3: read BBBABAB9B8[31:0] then write BBBABAB9B8[31:0] @0x8</td>
<td></td>
<td>3. @0x8 / BBBABAB9B8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@0xC / BFBEBCBDC</td>
<td>4: read BFBEBCBDC[31:0] then write BFBEBCBDC[31:0] @0xC</td>
<td></td>
<td>4. @0xC / BFBEBCBDC</td>
</tr>
</tbody>
</table>
Addressing AHB peripherals not supporting byte/half-word write transfers

When the DMA controller initiates an AHB byte or half-word write transfer, the data are duplicated on the unused lanes of the AHB master 32-bit data bus (HWDATA[31:0]).

When the AHB slave peripheral does not support byte or half-word write transfers and does not generate any error, the DMA controller writes the 32 HWDATA bits as shown in the two examples below:

- To write the half-word 0xABCD, the DMA controller sets the HWDATA bus to 0xABCDABCD with a half-word data size (HSIZE = HalfWord in AHB master bus).
- To write the byte 0xAB, the DMA controller sets the HWDATA bus to 0xABABABAB with a byte data size (HSIZE = Byte in the AHB master bus).

Assuming the AHB/APB bridge is an AHB 32-bit slave peripheral that does not take into account the HSIZE data, any AHB byte or half-word transfer is changed into a 32-bit APB transfer as described below:

- An AHB byte write transfer of 0xB0 to one of the 0x0, 0x1, 0x2 or 0x3 addresses, is converted to an APB word write transfer of 0xB0B0B0B0 to the 0x0 address.
- An AHB half-word write transfer of 0xB1B0 to the 0x0 or 0x2 addresses, is converted to an APB word write transfer of 0xB1B0B1B0 to the 0x0 address.

10.4.6 DMA error management

A DMA transfer error is generated when reading from or writing to a reserved address space. When a DMA transfer error occurs during a DMA read or write access, the faulty channel x is automatically disabled through a hardware clear of its EN bit in the corresponding DMA_CCRx register.

The TEIFx bit of the DMA_ISR register is set. An interrupt is then generated if the TEIE bit of the DMA_CCRx register is set.

The EN bit of the DMA_CCRx register can not be set again by software (channel x re-activated) until the TEIFx bit of the DMA_ISR register is cleared (by setting the CTEIFx bit of the DMA_IFCR register).

When the software is notified with a transfer error over a channel which involves a peripheral, the software has first to stop this peripheral in DMA mode, in order to disable any pending or future DMA request. Then software may normally reconfigure both DMA and the peripheral in DMA mode for a new transfer.
10.5 DMA interrupts

An interrupt can be generated on a half transfer, transfer complete or transfer error for each DMA channel x. Separate interrupt enable bits are available for flexibility.

Table 29. DMA interrupt requests

<table>
<thead>
<tr>
<th>Interrupt request</th>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Interrupt enable bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel x interrupt</td>
<td>Half transfer on channel x</td>
<td>HTIFx</td>
<td>HTIEx</td>
</tr>
<tr>
<td></td>
<td>Transfer complete on channel x</td>
<td>TCIFx</td>
<td>TCIEx</td>
</tr>
<tr>
<td></td>
<td>Transfer error on channel x</td>
<td>TEIFx</td>
<td>TEIEx</td>
</tr>
<tr>
<td></td>
<td>Half transfer or transfer complete or transfer error on channel x</td>
<td>GIFx</td>
<td>-</td>
</tr>
</tbody>
</table>

10.6 DMA registers

Refer to Section 1.2 for a list of abbreviations used in register descriptions.

The DMA registers have to be accessed by words (32-bit).

10.6.1 DMA interrupt status register (DMA_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

Every status bit is cleared by hardware when the software sets the corresponding clear bit or the corresponding global clear bit CGIFx, in the DMA_IFCR register.

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTIF4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCI4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEIF4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIF4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTIF3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCI3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEIF3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIF3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTIF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCI2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEIF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTIF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCI1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEIF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:20 Reserved, must be kept at reset value.

- Bit 19 **TEIF5**: Transfer error (TE) flag for channel 5
  0: No TE event
  1: A TE event occurred.

- Bit 18 **HTIF5**: Half transfer (HT) flag for channel 5
  0: No HT event
  1: An HT event occurred.

- Bit 17 **TCIF5**: Transfer complete (TC) flag for channel 5
  0: No TC event
  1: A TC event occurred.

- Bit 16 **GIF5**: global interrupt flag for channel 5
  0: No TE, HT, or TC event
  1: A TE, HT, or TC event occurred.
Bit 15 **TEIF4**: Transfer error (TE) flag for channel 4
0: No TE event
1: A TE event occurred.

Bit 14 **HTIF4**: Half transfer (HT) flag for channel 4
0: No HT event
1: An HT event occurred.

Bit 13 **TCIF4**: Transfer complete (TC) flag for channel 4
0: No TC event
1: A TC event occurred.

Bit 12 **GIF4**: Global interrupt flag for channel 4
0: No TE, HT, or TC event
1: A TE, HT, or TC event occurred.

Bit 11 **TEIF3**: Transfer error (TE) flag for channel 3
0: No TE event
1: A TE event occurred.

Bit 10 **HTIF3**: Half transfer (HT) flag for channel 3
0: No HT event
1: An HT event occurred.

Bit 9 **TCIF3**: Transfer complete (TC) flag for channel 3
0: No TC event
1: A TC event occurred.

Bit 8 **GIF3**: Global interrupt flag for channel 3
0: No TE, HT, or TC event
1: A TE, HT, or TC event occurred.

Bit 7 **TEIF2**: Transfer error (TE) flag for channel 2
0: No TE event
1: A TE event occurred.

Bit 6 **HTIF2**: Half transfer (HT) flag for channel 2
0: No HT event
1: An HT event occurred.

Bit 5 **TCIF2**: Transfer complete (TC) flag for channel 2
0: No TC event
1: A TC event occurred.

Bit 4 **GIF2**: Global interrupt flag for channel 2
0: No TE, HT, or TC event
1: A TE, HT, or TC event occurred.

Bit 3 **TEIF1**: Transfer error (TE) flag for channel 1
0: No TE event
1: A TE event occurred.

Bit 2 **HTIF1**: Half transfer (HT) flag for channel 1
0: No HT event
1: An HT event occurred.

Bit 1 **TCIF1**: Transfer complete (TC) flag for channel 1
0: No TC event
1: A TC event occurred.
10.6.2 DMA interrupt flag clear register (DMA_IFCR)

Address offset: 0x04
Reset value: 0x0000 0000

Setting the global clear bit CGIFx of the channel x in this DMA_IFCR register, causes the DMA hardware to clear the corresponding GIFx bit and any individual flag among TEIFx, HTIFx, TCIFx, in the DMA_ISR register.

Setting any individual clear bit among CTEIFx, CHTIFx, CTCIFx in this DMA_IFCR register, causes the DMA hardware to clear the corresponding individual flag and the global flag GIFx in the DMA_ISR register, provided that none of the two other individual flags is set.

Writing 0 into any flag clear bit has no effect.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-20</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>19</td>
<td>CGIF5: Global interrupt flag clear for channel 5</td>
</tr>
<tr>
<td>18</td>
<td>CHTIF5: Half transfer flag clear for channel 5</td>
</tr>
<tr>
<td>17</td>
<td>CTCIF5: Transfer complete flag clear for channel 5</td>
</tr>
<tr>
<td>16</td>
<td>CGIF4: Global interrupt flag clear for channel 4</td>
</tr>
<tr>
<td>15</td>
<td>CTEIF4: Transfer error flag clear for channel 4</td>
</tr>
<tr>
<td>14</td>
<td>CHTIF4: Half transfer flag clear for channel 4</td>
</tr>
<tr>
<td>13</td>
<td>CTCIF4: Transfer complete flag clear for channel 4</td>
</tr>
<tr>
<td>12</td>
<td>CGIF3: Global interrupt flag clear for channel 3</td>
</tr>
<tr>
<td>11</td>
<td>CTEIF3: Transfer error flag clear for channel 3</td>
</tr>
<tr>
<td>10</td>
<td>CHTIF3: Half transfer flag clear for channel 3</td>
</tr>
<tr>
<td>9</td>
<td>CTCIF3: Transfer complete flag clear for channel 3</td>
</tr>
<tr>
<td>8</td>
<td>CGIF2: Global interrupt flag clear for channel 2</td>
</tr>
<tr>
<td>7</td>
<td>CTEIF2: Transfer error flag clear for channel 2</td>
</tr>
<tr>
<td>6</td>
<td>CHTIF2: Half transfer flag clear for channel 2</td>
</tr>
<tr>
<td>5</td>
<td>CTCIF2: Transfer complete flag clear for channel 2</td>
</tr>
<tr>
<td>4</td>
<td>CGIF1: Global interrupt flag clear for channel 1</td>
</tr>
<tr>
<td>3</td>
<td>CTEIF1: Transfer error flag clear for channel 1</td>
</tr>
</tbody>
</table>
Bit 2 CHTIF1: Half transfer flag clear for channel 1
Bit 1 CTCIF1: Transfer complete flag clear for channel 1
Bit 0 CGIF1: Global interrupt flag clear for channel 1

10.6.3 DMA channel x configuration register (DMA_CCRx)

Address offset: 0x08 + 0x14 * (x - 1), (x = 1 to 5)
Reset value: 0x0000 0000

The register fields/bits MEM2MEM, PL[1:0], MSIZE[1:0], PSIZE[1:0], MINC, PINC, and DIR are read-only when EN = 1.

The states of MEM2MEM and CIRC bits must not be both high at the same time.

<table>
<thead>
<tr>
<th>Mem2MEM</th>
<th>PL[1:0]</th>
<th>MSIZE[1:0]</th>
<th>PSIZE[1:0]</th>
<th>MINC</th>
<th>PINC</th>
<th>CIRC</th>
<th>DIR</th>
<th>TEIE</th>
<th>HTIE</th>
<th>TCIE</th>
<th>EN</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:15 Reserved, must be kept at reset value.

Bit 14 MEM2MEM: Memory-to-memory mode
0: Disabled
1: Enabled

Note: This bit is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).

Bits 13:12 PL[1:0]: Priority level
00: Low
01: Medium
10: High
11: Very high

Note: This bitfield is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).

Bits 11:10 MSIZE[1:0]: Memory size
Defines the data size of each DMA transfer to the identified memory.
In memory-to-memory mode, this bitfield identifies the memory source if DIR = 1 and the memory destination if DIR = 0.
In peripheral-to-peripheral mode, this bitfield identifies the peripheral source if DIR = 1 and the peripheral destination if DIR = 0.
00: 8 bits
01: 16 bits
10: 32 bits
11: Reserved

Note: This bitfield is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).
Bits 9:8 **PSIZE[1:0]**: Peripheral size

Defines the data size of each DMA transfer to the identified peripheral.

- In memory-to-memory mode, this bitfield identifies the memory destination if DIR = 1 and the memory source if DIR = 0.
- In peripheral-to-peripheral mode, this bitfield identifies the peripheral destination if DIR = 1 and the peripheral source if DIR = 0.

- 00: 8 bits
- 01: 16 bits
- 10: 32 bits
- 11: Reserved

*Note: This bitfield is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).*

Bit 7 **MINC**: Memory increment mode

Defines the increment mode for each DMA transfer to the identified memory.

- In memory-to-memory mode, this bit identifies the memory source if DIR = 1 and the memory destination if DIR = 0.
- In peripheral-to-peripheral mode, this bit identifies the peripheral source if DIR = 1 and the peripheral destination if DIR = 0.

- 0: Disabled
- 1: Enabled

*Note: This bit is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).*

Bit 6 **PINC**: Peripheral increment mode

Defines the increment mode for each DMA transfer to the identified peripheral.

- In memory-to-memory mode, this bit identifies the memory destination if DIR = 1 and the memory source if DIR = 0.
- In peripheral-to-peripheral mode, this bit identifies the peripheral destination if DIR = 1 and the peripheral source if DIR = 0.

- 0: Disabled
- 1: Enabled

*Note: This bit is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).*

Bit 5 **CIRC**: Circular mode

- 0: Disabled
- 1: Enabled

*Note: This bit is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is not read-only when the channel is enabled (EN = 1).*
10.6.4 DMA channel x number of data to transfer register (DMA_CNDTRx)

Address offset: 0x0C + 0x14 * (x - 1), (x = 1 to 5)

Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDT[15:0]</td>
<td>Number of data to transfer</td>
</tr>
<tr>
<td>rw</td>
<td>Readable and writable</td>
</tr>
</tbody>
</table>

Note: This bit is set and cleared by software.
### Bits 31:16
Reserved, must be kept at reset value.

### Bits 15:0 **NDT[15:0]**: Number of data to transfer (0 to $2^{16} - 1$)

This bitfield is updated by hardware when the channel is enabled:
- It is decremented after each single DMA 'read followed by write' transfer, indicating the remaining amount of data items to transfer.
- It is kept at zero when the programmed amount of data to transfer is reached, if the channel is not in circular mode (CIRC = 0 in the DMA_CCRx register).
- It is reloaded automatically by the previously programmed value, when the transfer is complete, if the channel is in circular mode (CIRC = 1).

If this bitfield is zero, no transfer can be served whatever the channel status (enabled or not).

**Note:** This bitfield is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is read-only when the channel is enabled (EN = 1).

#### 10.6.5 DMA channel x peripheral address register (DMA_CPARx)

**Address offset:** $0x10 + 0x14 * (x - 1)$, ($x = 1$ to 5)

**Reset value:** $0x0000 0000$

<table>
<thead>
<tr>
<th>PA[31:16]</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PA[15:0]</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>9876543210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits 31:0 **PA[31:0]**: Peripheral address

It contains the base address of the peripheral data register from/to which the data is read/written.

When PSIZE[1:0] = 01 (16 bits), bit 0 of PA[31:0] is ignored. Access is automatically aligned to a half-word address.

When PSIZE[1:0] = 10 (32 bits), bits 1 and 0 of PA[31:0] are ignored. Access is automatically aligned to a word address.

In memory-to-memory mode, this bitfield identifies the memory destination address if DIR = 1 and the memory source address if DIR = 0.

In peripheral-to-peripheral mode, this bitfield identifies the peripheral destination address if DIR = 1 and the peripheral source address if DIR = 0.

**Note:** This bitfield is set and cleared by software. It must not be written when the channel is enabled (EN = 1). It is not read-only when the channel is enabled (EN = 1).
10.6.6 DMA channel x memory address register (DMA_CMARx)

Address offset: 0x14 + 0x14 * (x - 1), (x = 1 to 5)
Reset value: 0x0000 0000

![Address Map](image)

Bits 31:0 **MA[31:0]**: Peripheral address
- It contains the base address of the memory from/to which the data is read/written.
- When **MSIZE[1:0] = 01** (16 bits), bit 0 of **MA[31:0]** is ignored. Access is automatically aligned to a half-word address.
- When **MSIZE[1:0] = 10** (32 bits), bits 1 and 0 of **MA[31:0]** are ignored. Access is automatically aligned to a word address.
- In memory-to-memory mode, this bitfield identifies the memory source address if **DIR = 1** and the memory destination address if **DIR = 0**.
- In peripheral-to-peripheral mode, this bitfield identifies the peripheral source address if **DIR = 1** and the peripheral destination address if **DIR = 0**.

**Note:** This bitfield is set and cleared by software. It must not be written when the channel is enabled (**EN = 1**). It is not read-only when the channel is enabled (**EN = 1**).

10.6.7 DMA channel selection register (DMA_CSELR)

Address offset: 0xA8
Reset value: 0x0000 0000

This register is present only on STM32F030xC devices.

This register is used to manage the mapping of DMA channels as detailed in *Section 10.3.2: DMA request mapping*.

![Selection Map](image)

Bits 31:20 Reserved, must be kept at reset value.
Bits 19:16 **C5S[3:0]**: DMA channel 5 selection
Details available in *Section 10.3.2: DMA request mapping*

Bits 15:12 **C4S[3:0]**: DMA channel 4 selection
Details available in *Section 10.3.2: DMA request mapping*

Bits 11:8 **C3S[3:0]**: DMA channel 3 selection
Details available in *Section 10.3.2: DMA request mapping*
### 10.6.8 DMA register map

#### Table 30. DMA register map and reset values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>DMA_ISR</td>
<td>0x004</td>
<td>DMA_IFCR</td>
<td>0x008</td>
<td>DMA_CCR1</td>
<td>0x00C</td>
<td>DMA_CNDTR1</td>
<td>0x010</td>
<td>DMA_CPAR1</td>
<td>0x014</td>
<td>DMA_CMAR1</td>
<td>0x018</td>
<td>Reserved</td>
<td>0x01C</td>
<td>DMA_CCR2</td>
<td>0x020</td>
<td>DMA_CNDTR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000</td>
<td>DMA_ISR</td>
<td>31</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>31</td>
<td>28</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td>0x004</td>
<td>DMA_IFCR</td>
<td>30</td>
<td>27</td>
<td>24</td>
<td>21</td>
<td>18</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>27</td>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>0x008</td>
<td>DMA_CCR1</td>
<td>29</td>
<td>26</td>
<td>23</td>
<td>20</td>
<td>17</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>29</td>
<td>26</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>0x00C</td>
<td>DMA_CNDTR1</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>0x010</td>
<td>DMA_CPAR1</td>
<td>31</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>31</td>
<td>28</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td>0x014</td>
<td>DMA_CMAR1</td>
<td>30</td>
<td>27</td>
<td>24</td>
<td>21</td>
<td>18</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>27</td>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>0x018</td>
<td>Reserved</td>
<td>29</td>
<td>26</td>
<td>23</td>
<td>20</td>
<td>17</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>29</td>
<td>26</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>0x01C</td>
<td>DMA_CCR2</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>0x020</td>
<td>DMA_CNDTR2</td>
<td>27</td>
<td>24</td>
<td>21</td>
<td>18</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>27</td>
<td>24</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td>0x024</td>
<td>DMA_CPAR2</td>
<td>26</td>
<td>23</td>
<td>20</td>
<td>17</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>26</td>
<td>23</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>0x028</td>
<td>DMA_CMAR2</td>
<td>25</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>22</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>0x02C</td>
<td>Reserved</td>
<td>24</td>
<td>21</td>
<td>18</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>21</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>0x030</td>
<td>DMA_CCR3</td>
<td>23</td>
<td>20</td>
<td>17</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>20</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>0x034</td>
<td>DMA_CNDTR3</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>19</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>0x038</td>
<td>DMA_CPAR3</td>
<td>21</td>
<td>18</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>18</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>0x03C</td>
<td>DMA_CMAR3</td>
<td>20</td>
<td>17</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>17</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>0x040</td>
<td>Reserved</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>16</td>
<td>13</td>
<td>10</td>
</tr>
</tbody>
</table>

**Bits 7:4 C2S[3:0]: DMA channel 2 selection**
- Details available in Section 10.3.2: DMA request mapping

**Bits 3:0 C1S[3:0]: DMA channel 1 selection**
- Details available in Section 10.3.2: DMA request mapping
Refer to *Section 2.2* for the register boundary addresses.
11 Interrupts and events

11.1 Nested vectored interrupt controller (NVIC)

11.1.1 NVIC main features

- 32 maskable interrupt channels (not including the sixteen Arm® Cortex®-M0 interrupt lines)
- 4 programmable priority levels (2 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, refer to the PM0215 programming manual.

For code example refer to the Appendix section A.6.1: NVIC initialization.

11.1.2 SysTick calibration value register

The SysTick calibration value is set to 6000, which gives a reference time base of 1 ms with the SysTick clock set to 6 MHz (max f_{HCLK} / 8).

11.1.3 Interrupt and exception vectors

*Table 31* is the vector table for STM32F0x0 devices. Consider peripheral availability on your device.

**Table 31. Vector table**

<table>
<thead>
<tr>
<th>Position</th>
<th>Priority</th>
<th>Type of priority</th>
<th>Acronym</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>Reserved</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>-</td>
<td>-3</td>
<td>Fixed</td>
<td>Reset</td>
<td>Reset</td>
<td>0x0000 0004</td>
</tr>
<tr>
<td>-</td>
<td>-2</td>
<td>Fixed</td>
<td>NMI</td>
<td>Non maskable interrupt. The RCC clock security system (CSS) is linked to the NMI vector.</td>
<td>0x0000 0008</td>
</tr>
<tr>
<td>-</td>
<td>-1</td>
<td>Fixed</td>
<td>HardFault</td>
<td>All classes of fault</td>
<td>0x0000 000C</td>
</tr>
<tr>
<td>-</td>
<td>3</td>
<td>Settable</td>
<td>SVCall</td>
<td>System service call via SWI instruction</td>
<td>0x0000 002C</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>Settable</td>
<td>PendSV</td>
<td>Pendable request for system service</td>
<td>0x0000 0038</td>
</tr>
<tr>
<td>-</td>
<td>6</td>
<td>Settable</td>
<td>SysTick</td>
<td>System tick timer</td>
<td>0x0000 003C</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Settable</td>
<td>WWWDG</td>
<td>Window watchdog interrupt</td>
<td>0x0000 0040</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>-</td>
<td>0x0000 0044</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>Settable</td>
<td>RTC</td>
<td>RTC interrupts (combined EXTI lines 17, 19 and 20)</td>
<td>0x0000 0048</td>
</tr>
</tbody>
</table>
### Table 31. Vector table (continued)

<table>
<thead>
<tr>
<th>Position</th>
<th>Priority</th>
<th>Type of priority</th>
<th>Acronym</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>Settable</td>
<td>FLASH</td>
<td>Flash global interrupt</td>
<td>0x0000 004C</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>Settable</td>
<td>RCC</td>
<td>RCC global interrupts</td>
<td>0x0000 0050</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>Settable</td>
<td>EXTI0_1</td>
<td>EXTI Line[1:0] interrupts</td>
<td>0x0000 0054</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>Settable</td>
<td>EXTI2_3</td>
<td>EXTI Line[3:2] interrupts</td>
<td>0x0000 0058</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>Settable</td>
<td>EXTI4_15</td>
<td>EXTI Line[15:4] interrupts</td>
<td>0x0000 005C</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>-</td>
<td>0x0000 0060</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>Settable</td>
<td>DMA_CH1</td>
<td>DMA channel 1 interrupt</td>
<td>0x0000 0064</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>Settable</td>
<td>DMA_CH2_3</td>
<td>DMA channel 2 and 3 interrupts</td>
<td>0x0000 0068</td>
</tr>
<tr>
<td>11</td>
<td>18</td>
<td>Settable</td>
<td>DMA_CH4_5</td>
<td>DMA channel 4 and 5 interrupts</td>
<td>0x0000 006C</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
<td>Settable</td>
<td>ADC</td>
<td>ADC interrupts</td>
<td>0x0000 0070</td>
</tr>
<tr>
<td>13</td>
<td>20</td>
<td>Settable</td>
<td>TIM1_BRK_UP_-TRG_COM</td>
<td>TIM1 break, update, trigger and commutation interrupt</td>
<td>0x0000 0074</td>
</tr>
<tr>
<td>14</td>
<td>21</td>
<td>Settable</td>
<td>TIM1_CC</td>
<td>TIM1 capture compare interrupt</td>
<td>0x0000 0078</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>-</td>
<td>0x0000 007C</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>Settable</td>
<td>TIM3</td>
<td>TIM3 global interrupt</td>
<td>0x0000 0080</td>
</tr>
<tr>
<td>17</td>
<td>24</td>
<td>Settable</td>
<td>TIM6</td>
<td>TIM6 global interrupt</td>
<td>0x0000 0084</td>
</tr>
<tr>
<td>18</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>-</td>
<td>0x0000 0088</td>
</tr>
<tr>
<td>19</td>
<td>26</td>
<td>Settable</td>
<td>TIM14</td>
<td>TIM14 global interrupt</td>
<td>0x0000 008C</td>
</tr>
<tr>
<td>20</td>
<td>27</td>
<td>Settable</td>
<td>TIM15</td>
<td>TIM15 global interrupt</td>
<td>0x0000 0090</td>
</tr>
<tr>
<td>21</td>
<td>28</td>
<td>Settable</td>
<td>TIM16</td>
<td>TIM16 global interrupt</td>
<td>0x0000 0094</td>
</tr>
<tr>
<td>22</td>
<td>29</td>
<td>Settable</td>
<td>TIM17</td>
<td>TIM17 global interrupt</td>
<td>0x0000 0098</td>
</tr>
<tr>
<td>23</td>
<td>30</td>
<td>Settable</td>
<td>I2C1</td>
<td>I²C1 global interrupt</td>
<td>0x0000 009C</td>
</tr>
<tr>
<td>24</td>
<td>31</td>
<td>Settable</td>
<td>I2C2</td>
<td>I²C2 global interrupt</td>
<td>0x0000 00A0</td>
</tr>
<tr>
<td>25</td>
<td>32</td>
<td>Settable</td>
<td>SPI1</td>
<td>SPI1 global interrupt</td>
<td>0x0000 00A4</td>
</tr>
<tr>
<td>26</td>
<td>33</td>
<td>Settable</td>
<td>SPI2</td>
<td>SPI2 global interrupt</td>
<td>0x0000 00A8</td>
</tr>
<tr>
<td>27</td>
<td>34</td>
<td>Settable</td>
<td>USART1</td>
<td>USART1 global interrupt</td>
<td>0x0000 00AC</td>
</tr>
<tr>
<td>28</td>
<td>35</td>
<td>Settable</td>
<td>USART2</td>
<td>USART2 global interrupt</td>
<td>0x0000 00B0</td>
</tr>
<tr>
<td>29</td>
<td>36</td>
<td>Settable</td>
<td>USART3_4_5_6</td>
<td>USART3, USART4, USART5, USART6 global interrupts</td>
<td>0x0000 00B4</td>
</tr>
<tr>
<td>30</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>-</td>
<td>0x0000 00B8</td>
</tr>
<tr>
<td>31</td>
<td>38</td>
<td>Settable</td>
<td>USB</td>
<td>USB global interrupt (combined with EXTI line 18)</td>
<td>0x0000 00BC</td>
</tr>
</tbody>
</table>
11.2 Extended interrupts and events controller (EXTI)

The extended interrupts and events controller (EXTI) manages the external and internal asynchronous events/interrupts and generates the event request to the CPU/Interrupt controller and a wake-up request to the Power manager.

The EXTI allows the management of up to 28 external/internal event line (21 external event lines and 7 internal event lines).

The active edge of each external interrupt line can be chosen independently, whilst for internal interrupt the active edge is always the rising one. An interrupt could be left pending: in case of an external one, a status register is instantiated and indicates the source of the interrupt; an event is always a simple pulse and it’s used for triggering the core Wake-up (e.g. Cortex-M0 RXEV pin). For internal interrupts, the pending status is assured by the generating IP, so no need for a specific flag. Each input line can be masked independently for interrupt or event generation, in addition the internal lines are sampled only in STOP mode. This controller allows also to emulate the (only) external events by software, multiplexed with the corresponding hardware event line, by writing to a dedicated register.

11.2.1 Main features

The EXTI main features are the following:

- Supports generation of up to 32 event/interrupt requests
- Independent mask on each event/interrupt line
- Automatic disable of internal lines when system is not in STOP mode
- Independent trigger for external event/interrupt line
- Dedicated status bit for external interrupt line
- Emulation for all the external event requests
11.2.2 Block diagram

The extended interrupt/event block diagram is shown in Figure 21.

Figure 21. Extended interrupts and events controller (EXTI) block diagram

11.2.3 Event management

The STM32F0x0 is able to handle external or internal events in order to wake up the core (WFE). The wake-up 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-M0 System control register. When the MCU resumes from WFE, the EXTI peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.
- or by 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.

11.2.4 Functional description

For the external interrupt lines, 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.
For the internal interrupt lines, the active edge is always the rising edge, the interrupt is enabled by default in the interrupt mask register and there is no corresponding pending bit 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.

For the external lines, an interrupt/event request can also be generated by software by writing a ‘1’ in the software interrupt/event register.

*Note:* The interrupts or events associated to the internal lines can be triggered only when the system is in STOP mode. If the system is still running, no interrupt/event is generated.

For code example refer to the Appendix section A.6.2: External interrupt selection.

**Hardware interrupt selection**

To configure a line as interrupt source, use the following procedure:

- Configure the corresponding mask bit in the EXTI_IMR register.
- Configure the trigger selection bits of the interrupt line (EXTI_RTSR and EXTI_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the EXTI so that an interrupt coming from one of the EXTI line can be correctly acknowledged.

**Hardware event selection**

To configure a line as event source, use the following procedure:

- Configure the corresponding mask bit in the EXTI_EMR register.
- Configure the Trigger Selection bits of the Event line (EXTI_RTSR and EXTI_FTSR)

**Software interrupt/event selection**

Any of the external lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

- Configure the corresponding mask bit (EXTI_IMR, EXTI_EMR)
- Set the required bit of the software interrupt register (EXTI_SWIER)
11.2.5 External and internal interrupt/event line mapping

The GPIOs are connected to the 16 external interrupt/event lines in the following manner:

**Figure 22. External interrupt/event GPIO mapping**

The remaining lines are connected as follow:
- EXTI line 16 is reserved (internally held low)
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the internal USB wake-up event
- EXTI line 19 is connected to the RTC Tamper and TimeStamp events
- EXTI line 20 is connected to the RTC Wake-up event (available only on STM32F070xB and STM32F030xC devices)
- EXTI line 21 is reserved (internally held low)
- EXTI line 22 is reserved (internally held low)
- EXTI line 23 is reserved (internally held low)
- EXTI line 24 is reserved (internally held low)
- EXTI line 25 is reserved (internally held low)
- EXTI line 26 is reserved (internally held low)
- EXTI line 27 is reserved (internally held low)
- EXTI line 28 is reserved (internally held low)
- EXTI line 29 is reserved (internally held low)
- EXTI line 30 is reserved (internally held low)
- EXTI line 31 is reserved (internally held low)

**Note:** EXTI lines which are reserved or not used on some devices are considered as internal.
11.3 EXTI registers

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32-bit).

11.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00
Reset value: 0x0FF4 0000 (STM32F030x4, STM32F030x6 devices)
0x7FF4 0000 (STM32F070x6 devices)
0x0F94 0000 (STM32F030x8 devices)
0x7F84 0000 (STM32F070xB and STM32F030xC devices)

Note: The reset value for the internal lines is set to ‘1’ in order to enable the interrupt by default.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM31</td>
<td>IM30</td>
<td>IM29</td>
<td>IM28</td>
<td>IM27</td>
<td>IM26</td>
<td>IM25</td>
<td>IM24</td>
<td>IM23</td>
<td>IM22</td>
<td>IM21</td>
<td>IM20</td>
<td>IM19</td>
<td>IM18</td>
<td>IM17</td>
<td>IM16</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>IM15</td>
<td>IM14</td>
<td>IM13</td>
<td>IM12</td>
<td>IM11</td>
<td>IM10</td>
<td>IM9</td>
<td>IM8</td>
<td>IM7</td>
<td>IM6</td>
<td>IM5</td>
<td>IM4</td>
<td>IM3</td>
<td>IM2</td>
<td>IM1</td>
<td>IM0</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:0 IMx: Interrupt Mask on line x (x = 31 to 0)
0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked

11.3.2 Event mask register (EXTI_EMR)

Address offset: 0x04
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>EM15</td>
<td>EM14</td>
<td>EM13</td>
<td>EM12</td>
<td>EM11</td>
<td>EM10</td>
<td>EM9</td>
<td>EM8</td>
<td>EM7</td>
<td>EM6</td>
<td>EM5</td>
<td>EM4</td>
<td>EM3</td>
<td>EM2</td>
<td>EM1</td>
<td>EM0</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:0 EMx: Event mask on line x (x = 31 to 0)
0: Event request from Line x is masked
1: Event request from Line x is not masked

11.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT22</td>
<td>RT21</td>
<td>RT20</td>
<td>RT19</td>
<td>Res.</td>
<td>RT17</td>
<td>RT16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interrupts and events

11.3.4 Falling trigger selection register (EXTI_FTSR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>FT31: Falling trigger event configuration bit of line 31</td>
<td>rw</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>30-23</td>
<td>Reserved</td>
<td>rw</td>
<td>Reserved</td>
</tr>
<tr>
<td>22-19</td>
<td>FTx: Falling trigger event configuration bit of line x (x = 22 to 19)</td>
<td>rw</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
<td>rw</td>
<td>Reserved</td>
</tr>
<tr>
<td>17-0</td>
<td>FTx: Falling trigger event configuration bit of line x (x = 17 to 0)</td>
<td>rw</td>
<td>0x0000 0000</td>
</tr>
</tbody>
</table>

Bit 31  **FT31**: Falling trigger event configuration bit of line 31
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

Bits 30:23  Reserved, must be kept at reset value.

Bits 22:19  **FTx**: Falling trigger event configuration bit of line x (x = 22 to 19)
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

Bit 18  Reserved, must be kept at reset value.

Bits 17:0  **FTx**: Falling trigger event configuration bit of line x (x = 17 to 0)
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 wake-up lines are edge triggered. No glitches must be generated on these lines. If a rising edge on an external interrupt line occurs during a write operation to the EXTI_RTSR register, the pending bit is not set.

Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
Note: The external wake-up lines are edge triggered. No glitches must be generated on these lines. If a falling edge on an external interrupt line occurs during a write operation to the EXTI_FTSR register, the pending bit is not set.
Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

11.3.5 Software interrupt event register (EXTI_SWIER)
Address offset: 0x10
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>SWI31: Software interrupt on line 31</th>
</tr>
</thead>
</table>
|        | If the interrupt is enabled on this line in the EXTI_IMR, writing a ‘1’ to this bit when it is at ‘0’ sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.
|        | This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1’ to the bit) |

| Bits 30:23 | Reserved, must be kept at reset value. |

| Bits 22:19 | SWIx: Software interrupt on line x (x = 22 to 19) |
|           | If the interrupt is enabled on this line in the EXTI_IMR, writing a ‘1’ to this bit when it is at ‘0’ sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. |
|           | This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1’ to the bit) |

| Bit 18 | Reserved, must be kept at reset value. |

| Bits 17:0 | SWIx: Software interrupt on line x (x = 17 to 0) |
|           | If the interrupt is enabled on this line in the EXTI_IMR, writing a ‘1’ to this bit when it is at ‘0’ sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. |
|           | This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1’ to the bit). |

11.3.6 Pending register (EXTI_PR)
Address offset: 0x14
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>PIF31: Pending interrupt on line 31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If the interrupt is enabled on this line in the EXTI_IMR, writing a ‘1’ to this bit when it is at ‘0’ sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.</td>
</tr>
<tr>
<td></td>
<td>This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1’ to the bit)</td>
</tr>
</tbody>
</table>

| Bits 30:23 | Reserved, must be kept at reset value. |

| Bits 22:19 | PIFx: Pending interrupt on line x (x = 22 to 19) |
|           | If the interrupt is enabled on this line in the EXTI_IMR, writing a ‘1’ to this bit when it is at ‘0’ sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. |
|           | This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1’ to the bit) |

| Bit 18 | Reserved, must be kept at reset value. |

| Bits 17:0 | PIFx: Pending interrupt on line x (x = 17 to 0) |
|           | If the interrupt is enabled on this line in the EXTI_IMR, writing a ‘1’ to this bit when it is at ‘0’ sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. |
|           | This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1’ to the bit). |
Bit 31  **PIF31**: Pending bit on line 31  
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 to the bit.

Bits 30:23  Reserved, must be kept at reset value.

Bits 22:19  **PIFx**: Pending bit on line x (x = 22 to 19)  
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 to the bit.

Bit 18  Reserved, must be kept at reset value.

Bits 17:0  **PIFx**: Pending bit on line x (x = 17 to 0)  
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 to the bit.
### 11.3.7 EXTI register map

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

**Table 32. External interrupt/event controller register map and reset values**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Offset</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>0x04</td>
<td>EXTI_EMR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>0x08</td>
<td>EXTI_RTSR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0C</td>
<td>EXTI_FTSR</td>
</tr>
<tr>
<td>0x10</td>
<td>EXTI_SWIER</td>
<td>0x14</td>
<td>EXTI_PR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Offset</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>0x04</td>
<td>EXTI_EMR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>0x08</td>
<td>EXTI_RTSR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0C</td>
<td>EXTI_FTSR</td>
</tr>
<tr>
<td>0x10</td>
<td>EXTI_SWIER</td>
<td>0x14</td>
<td>EXTI_PR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Offset</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>0x04</td>
<td>EXTI_EMR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>0x08</td>
<td>EXTI_RTSR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0C</td>
<td>EXTI_FTSR</td>
</tr>
<tr>
<td>0x10</td>
<td>EXTI_SWIER</td>
<td>0x14</td>
<td>EXTI_PR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refer to *Section 2.2 on page 37* for the register boundary addresses.
12  Analog-to-digital converter (ADC)

12.1 Introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it to measure signals from 16 external and 2 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 higher or lower thresholds.

An efficient low-power mode is implemented to allow very low consumption at low frequency.
### 12.2 ADC main features

- **High performance**
  - 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
  - ADC conversion time: 1.0 µs for 12-bit resolution (1 MHz), 0.93 µs conversion time for 10-bit resolution, faster conversion times can be obtained by lowering resolution.
  - Self-calibration
  - Programmable sampling time
  - Data alignment with built-in data coherency
  - DMA support

- **Low-power**
  - The application can reduce PCLK frequency for low-power operation while still keeping optimum ADC performance. For example, 1.0 µs conversion time is kept, whatever the PCLK frequency
  - Wait mode: prevents ADC overrun in applications with low PCLK frequency
  - Auto off mode: ADC is automatically powered off except during the active conversion phase. This dramatically reduces the power consumption of the ADC.

- **Analog input channels**
  - 16 external analog inputs
  - 1 channel for internal temperature sensor (VSENSE)
  - 1 channel for internal reference voltage (VREFINT)

- **Start-of-conversion can be initiated:**
  - By software
  - By hardware triggers with configurable polarity (timer events)

- **Conversion modes**
  - Can convert a single channel or can scan a sequence of channels.
  - Single mode converts selected inputs once per trigger
  - Continuous mode converts selected inputs continuously
  - Discontinuous mode

- **Interrupt generation at the end of sampling, end of conversion, end of sequence conversion, and in case of analog watchdog or overrun events**

- **Analog watchdog**

- **ADC input range:** $V_{SSA} \leq V_{IN} \leq V_{DDA}$
12.3 ADC functional description

*Figure 23* shows the ADC block diagram and *Table 33* gives the ADC pin description.

**Figure 23. ADC block diagram**

#### 12.3.1 ADC pins and internal signals

**Table 33. ADC input/output pins**

<table>
<thead>
<tr>
<th>Name</th>
<th>Signal type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDDA</td>
<td>Input, analog power supply</td>
<td>Analog power supply and positive reference voltage for the ADC</td>
</tr>
<tr>
<td>VSSA</td>
<td>Input, analog supply ground</td>
<td>Ground for analog power supply</td>
</tr>
<tr>
<td>ADC_INx</td>
<td>Analog input signals</td>
<td>16 external analog input channels</td>
</tr>
</tbody>
</table>
12.3.2 Calibration (ADCAL)

The ADC has a calibration feature. During the calibration phase, the ADC calculates a calibration factor which is internally applied to the ADC until the next ADC power-off. The application must not use the ADC during calibration and must wait until it is complete.

Calibration should be performed before starting A/D conversion. It removes the offset error which may vary from chip to chip due to process variation.

The calibration is initiated by software by setting ADCAL bit to 1. It can only be initiated when the ADC is disabled (when ADEN = 0). ADCAL bit stays at 1 during the whole calibration sequence. It is then cleared by hardware as soon the calibration completes. After this, the calibration factor can be read from the ADC_DR register (from bits 6 to 0).

The internal analog calibration is kept if the ADC is disabled (ADEN = 0). When the ADC operating conditions change (V_{DDA} changes are the main contributor to ADC offset variations and temperature change to a lesser extend), it is recommended to re-run a calibration cycle.

The calibration factor is lost each time power is removed from the ADC (for example when the product enters Standby mode).

### Table 34. ADC internal input/output signals

<table>
<thead>
<tr>
<th>Internal signal name</th>
<th>Signal type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{IN}[x]</td>
<td>Analog Input</td>
<td>Connected either to internal channels or to ADC_IN[i] external channels</td>
</tr>
<tr>
<td>TRGx</td>
<td>Input</td>
<td>ADC conversion triggers</td>
</tr>
<tr>
<td>V_{SENSE}</td>
<td>Input</td>
<td>Internal temperature sensor output voltage</td>
</tr>
<tr>
<td>V_{REFINT}</td>
<td>Input</td>
<td>Internal voltage reference output voltage</td>
</tr>
<tr>
<td>ADC_AWDx_OUT</td>
<td>Output</td>
<td>Internal analog watchdog output signal connected to on-chip timers (x = Analog watchdog number = 1)</td>
</tr>
</tbody>
</table>

### Table 35. External triggers

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>EXTSEL[2:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRG0</td>
<td>TIM1_TRGO</td>
<td>000</td>
</tr>
<tr>
<td>TRG1</td>
<td>TIM1_CC4</td>
<td>001</td>
</tr>
<tr>
<td>TRG2</td>
<td>TIM2_TRGO</td>
<td>010</td>
</tr>
<tr>
<td>TRG3</td>
<td>TIM3_TRGO</td>
<td>011</td>
</tr>
<tr>
<td>TRG4</td>
<td>TIM15_TRGO</td>
<td>100</td>
</tr>
<tr>
<td>TRG5</td>
<td>Reserved</td>
<td>101</td>
</tr>
<tr>
<td>TRG6</td>
<td>Reserved</td>
<td>110</td>
</tr>
<tr>
<td>TRG7</td>
<td>Reserved</td>
<td>111</td>
</tr>
</tbody>
</table>
Calibration software procedure
1. Ensure that ADEN = 0 and DMAEN = 0.
2. Set ADCAL = 1.
3. Wait until ADCAL = 0.
4. The calibration factor can be read from bits 6:0 of ADC_DR.
For code example refer to the Appendix section A.7.1: ADC calibration.

![Figure 24. ADC calibration](image)

12.3.3 ADC on-off control (ADEN, ADDIS, ADRDY)

At power-up, the ADC is disabled and put in power-down mode (ADEN = 0).

As shown in Figure 25, the ADC needs a stabilization time of $t_{\text{STAB}}$ before it starts converting accurately.

Two control bits are used to enable or disable the ADC:
- Set ADEN = 1 to enable the ADC. The ADRDY flag is set as soon as the ADC is ready for operation.
- Set ADDIS = 1 to disable the ADC and put the ADC in power down mode. The ADEN and ADDIS bits are then automatically cleared by hardware as soon as the ADC is fully disabled.

Conversion can then start either by setting ADSTART to 1 (refer to Section 12.4: Conversion on external trigger and trigger polarity (EXTSEL, EXTEN) on page 193) or when an external trigger event occurs if triggers are enabled.

Follow this procedure to enable the ADC:
1. Clear the ADRDY bit in ADC_ISR register by programming this bit to 1.
2. Set ADEN = 1 in the ADC_CR register.
3. Wait until ADRDY = 1 in the ADC_ISR register and continue to write ADEN = 1 (ADRDY is set after the ADC startup time). This can be handled by interrupt if the interrupt is enabled by setting the ADRDYIE bit in the ADC_IER register.

For code example refer to the Appendix section A.7.2: ADC enable sequence.
Follow this procedure to disable the ADC:

1. Check that ADSTART = 0 in the ADC_CR register to ensure that no conversion is ongoing. If required, stop any ongoing conversion by writing 1 to the ADSTP bit in the ADC_CR register and waiting until this bit is read at 0.
2. Set ADDIS = 1 in the ADC_CR register.
3. If required by the application, wait until ADEN = 0 in the ADC_CR register, indicating that the ADC is fully disabled (ADDIS is automatically reset once ADEN = 0).
4. Clear the ADRDY bit in ADC_ISR register by programming this bit to 1 (optional).

For code example refer to the Appendix section A.7.3: ADC disable sequence.

**Caution:** ADEN bit cannot be set when ADCAL = 1 and during four ADC clock cycles after the ADCAL bit is cleared by hardware (end of calibration).

---

**Figure 25. Enabling/disabling the ADC**

Note: In Auto-off mode (AUTOFF = 1) the power-on/off phases are performed automatically, by hardware and the ADRDY flag is not set.

When the bus clock is much faster than the analog ADC_CK clock, a minimum delay of ten analog ADC_CK cycles must be respected between ADEN and ADDIS bit settings.
### 12.3.4 ADC clock (CKMODE)

The ADC has a dual clock-domain architecture, so that the ADC can be fed with a clock
(ADC asynchronous clock) independent from the APB clock (PCLK).

**Figure 26. ADC clock scheme**

1. Refer to Section Reset and clock control (RCC) for how the PCLK clock and ADC asynchronous clock are enabled.

The input clock of the analog ADC can be selected between two different clock sources (see Figure 26: ADC clock scheme to see how the PCLK clock and the ADC asynchronous clock are enabled):

   a) The ADC clock can be a specific clock source, named “ADC asynchronous clock” which is independent and asynchronous with the APB clock.

   Refer to RCC Section for more information on generating this clock source.

   To select this scheme, bits CKMODE[1:0] of the ADC_CFGR2 register must be reset.

   For code example refer to the Appendix section A.7.4: ADC clock selection.

   b) The ADC clock can be derived from the APB clock of the ADC bus interface, divided by a programmable factor (1, 2 or 4) according to bits CKMODE[1:0].

   To select this scheme, bits CKMODE[1:0] of the ADC_CFGR2 register must be different from “00”.

Option a) has the advantage of reaching the maximum ADC clock frequency whatever the APB clock scheme selected.

Option b) has the advantage of bypassing the clock domain resynchronizations. This can be useful when the ADC is triggered by a timer and if the application requires that the ADC is precisely triggered without any uncertainty (otherwise, an uncertainty of the trigger instant is added by the resynchronizations between the two clock domains).
12.3.5 Configuring the ADC

The software must write the ADCAL and ADEN bits in the ADC_CR register only when the ADC is disabled (ADEN cleared).

The software must only write to the ADSTART and ADDIS bits in the ADC_CR register only if the ADC is enabled and there is no pending request to disable the ADC (ADEN = 1 and ADDIS = 0).

For all the other control bits in the ADC_IER, ADC_CFGRi, ADC_SMPR, ADC_TR, ADC_CHSELR and ADC_CCR registers, refer to the description of the corresponding control bit in Section 12.10: ADC registers.

The software must only write to the ADSTP bit in the ADC_CR register if the ADC is enabled (and possibly converting) and there is no pending request to disable the ADC (ADSTART = 1 and ADDIS = 0).

Note: There is no hardware protection preventing software from making write operations forbidden by the above rules. If such a forbidden write access occurs, the ADC may enter an undefined state. To recover correct operation in this case, the ADC must be disabled (clear ADEN = 0 and all the bits in the ADC_CR register).

12.3.6 Channel selection (CHSEL, SCANDIR)

There are up to 18 multiplexed channels:
- 16 analog inputs from GPIO pins (ADC_INx)
- 2 internal analog inputs (temperature sensor, internal reference voltage)

It is possible to convert a single channel or a sequence of channels.

The sequence of the channels to be converted can be programmed in the ADC_CHSELR channel selection register: each analog input channel has a dedicated selection bit (CHSELe).

The order in which the channels is scanned can be configured by programming the bit SCANDIR bit in the ADC_CFGR1 register:
- SCANDIR = 0: forward scan Channel 0 to Channel 17
- SCANDIR = 1: backward scan Channel 17 to Channel 0

Temperature sensor, VREFINT internal channels

The temperature sensor is connected to channel ADC VIN[16].

### Table 36. Latency between trigger and start of conversion(1)

<table>
<thead>
<tr>
<th>ADC clock source</th>
<th>CKMODE[1:0]</th>
<th>Latency between the trigger event and the start of conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated 14MHz clock</td>
<td>00</td>
<td>Latency is not deterministic (jitter)</td>
</tr>
<tr>
<td>PCLK divided by 2</td>
<td>01</td>
<td>Latency is deterministic (no jitter) and equal to 2.75 ADC clock cycles</td>
</tr>
<tr>
<td>PCLK divided by 4</td>
<td>10</td>
<td>Latency is deterministic (no jitter) and equal to 2.625 ADC clock cycles</td>
</tr>
</tbody>
</table>

1. Refer to the device datasheet for the maximum ADC_CLK frequency.
The internal voltage reference $V_{\text{REFINT}}$ is connected to channel ADC $V_{\text{IN}[17]}$.

### 12.3.7 Programmable sampling time (SMP)

Before starting a conversion, the ADC needs to establish a direct connection between the voltage source to be measured and the embedded sampling capacitor of the ADC. This sampling time must be enough for the input voltage source to charge the sample and hold capacitor to the input voltage level.

Having a programmable sampling time allows the conversion speed to be trimmed according to the input resistance of the input voltage source.

The ADC samples the input voltage for a number of ADC clock cycles that can be modified using the SMP[2:0] bits in the ADC_SMPR register.

This programmable sampling time is common to all channels. If required by the application, the software can change and adapt this sampling time between each conversions.

The total conversion time is calculated as follows:

$$t_{\text{CONV}} = \text{Sampling time} + 12.5 \times \text{ADC clock cycles}$$

Example:

With ADC_CLK = 14 MHz and a sampling time of 1.5 ADC clock cycles:

$$t_{\text{CONV}} = 1.5 + 12.5 = 14 \text{ ADC clock cycles} = 1 \mu\text{s}$$

The ADC indicates the end of the sampling phase by setting the EOSMP flag.

### 12.3.8 Single conversion mode (CONT = 0)

In Single conversion mode, the ADC performs a single sequence of conversions, converting all the channels once. This mode is selected when CONT = 0 in the ADC_CFGR1 register. Conversion is started by either:

- Setting the ADSTART bit in the ADC_CR register
- Hardware trigger event

Inside the sequence, after each conversion is complete:

- The converted data are stored in the 16-bit ADC_DR register
- The EOC (end of conversion) flag is set
- An interrupt is generated if the EOCIE bit is set

After the sequence of conversions is complete:

- The EOS (end of sequence) flag is set
- An interrupt is generated if the EOSIE bit is set

Then the ADC stops until a new external trigger event occurs or the ADSTART bit is set again.

**Note:** To convert a single channel, program a sequence with a length of 1.
12.3.9 Continuous conversion mode (CONT = 1)

In continuous conversion mode, when a software or hardware trigger event occurs, the ADC performs a sequence of conversions, converting all the channels once and then automatically re-starts and continuously performs the same sequence of conversions. This mode is selected when CONT = 1 in the ADC_CFRG1 register. Conversion is started by either:

- Setting the ADSTART bit in the ADC_CR register
- Hardware trigger event

Inside the sequence, after each conversion is complete:

- The converted data are stored in the 16-bit ADC_DR register
- The EOC (end of conversion) flag is set
- An interrupt is generated if the EOCIE bit is set

After the sequence of conversions is complete:

- The EOS (end of sequence) flag is set
- An interrupt is generated if the EOSIE bit is set

Then, a new sequence restarts immediately and the ADC continuously repeats the conversion sequence.

Note: To convert a single channel, program a sequence with a length of 1.
It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.

12.3.10 Starting conversions (ADSTART)

Software starts ADC conversions by setting ADSTART = 1. When ADSTART is set, the conversion:

- Starts immediately if EXTEN = 00 (software trigger)
- At the next active edge of the selected hardware trigger if EXTEN ≠ 00

The ADSTART bit is also used to indicate whether an ADC operation is currently ongoing. It is possible to re-configure the ADC while ADSTART = 0, indicating that the ADC is idle.

The ADSTART bit is cleared by hardware:

- In single mode with software trigger (CONT = 0, EXTEN = 00)
  - At any end of conversion sequence (EOS = 1)
- In discontinuous mode with software trigger (CONT = 0, DISCEN = 1, EXTEN = 00)
  - At end of conversion (EOC = 1)
- In all cases (CONT = x, EXTEN = XX)
  - After execution of the ADSTP procedure invoked by software (see Section 12.3.12: Stopping an ongoing conversion (ADSTP) on page 193)

Note: In continuous mode (CONT = 1), the ADSTART bit is not cleared by hardware when the EOS flag is set because the sequence is automatically relaunched.

When hardware trigger is selected in single mode (CONT = 0 and EXTEN = 01), ADSTART is not cleared by hardware when the EOS flag is set (except if DMAEN = 1 and DMACFG = 0 in which case ADSTART is cleared at end of the DMA transfer). This avoids
the need for software having to set the ADSTART bit again and ensures the next trigger event is not missed.

12.3.11 Timings

The elapsed time between the start of a conversion and the end of conversion is the sum of the configured sampling time plus the successive approximation time depending on data resolution:

\[
t_{\text{CONV}} = t_{\text{SMPL}} + t_{\text{SAR}} = [1.5 \text{ } \mu \text{s} + 12.5 \text{ } \mu \text{s}] \times t_{\text{ADC_CLK}}
\]

\[
t_{\text{CONV}} = t_{\text{SMPL}} + t_{\text{SAR}} = 107.1 \text{ } \mu \text{s} + 892.8 \text{ } \mu \text{s} = 1 \text{ } \mu \text{s} \text{ (for } f_{\text{ADC_CLK}} = 14 \text{ MHz)}
\]

**Figure 27. Analog to digital conversion time**

**Figure 28. ADC conversion timings**

1. EXTEN = 00 or EXTEN ≠ 00
2. Trigger latency (refer to datasheet for more details)
3. ADC_DR register write latency (refer to datasheet for more details)
12.3.12 Stopping an ongoing conversion (ADSTP)

The software can decide to stop any ongoing conversions by setting ADSTP = 1 in the ADC_CR register.

This resets the ADC operation and the ADC is idle, ready for a new operation.

When the ADSTP bit is set by software, any ongoing conversion is aborted and the result is discarded (ADC_DR register is not updated with the current conversion).

The scan sequence is also aborted and reset (meaning that restarting the ADC would restart a new sequence).

Once this procedure is complete, the ADSTP and ADSTART bits are both cleared by hardware and the software must wait until ADSTART=0 before starting new conversions.

Figure 29. Stopping an ongoing conversion

12.4 Conversion on external trigger and trigger polarity (EXTSEL, EXTEN)

A conversion or a sequence of conversion can be triggered either by software or by an external event (for example timer capture). If the EXTEN[1:0] control bits are not equal to “0b00”, then external events are able to trigger a conversion with the selected polarity. The trigger selection is effective once software has set bit ADSTART = 1.

Any hardware triggers which occur while a conversion is ongoing are ignored.

If bit ADSTART = 0, any hardware triggers which occur are ignored.

Table 37 provides the correspondence between the EXTEN[1:0] values and the trigger polarity.

Table 37. Configuring the trigger polarity

<table>
<thead>
<tr>
<th>Source</th>
<th>EXTEN[1:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger detection disabled</td>
<td>00</td>
</tr>
<tr>
<td>Detection on rising edge</td>
<td>01</td>
</tr>
<tr>
<td>Detection on falling edge</td>
<td>10</td>
</tr>
<tr>
<td>Detection on both rising and falling edges</td>
<td>11</td>
</tr>
</tbody>
</table>

Note: The polarity of the external trigger can be changed only when the ADC is not converting (ADSTART = 0).

The EXTSEL[2:0] control bits are used to select which of 8 possible events can trigger conversions.
Refer to Table 35: External triggers in Section 12.3.1: ADC pins and internal signals for the list of all the external triggers that can be used for regular conversion.

The software source trigger events can be generated by setting the ADSTART bit in the ADC_CR register.

Note: The trigger selection can be changed only when the ADC is not converting (ADSTART = 0).

12.4.1 Discontinuous mode (DISCEN)

This mode is enabled by setting the DISCEN bit in the ADC_CFGR1 register.

In this mode (DISCEN = 1), a hardware or software trigger event is required to start each conversion defined in the sequence. On the contrary, if DISCEN = 0, a single hardware or software trigger event successively starts all the conversions defined in the sequence.

Example:

- DISCEN = 1, channels to be converted = 0, 3, 7, 10
  - 1st trigger: channel 0 is converted and an EOC event is generated
  - 2nd trigger: channel 3 is converted and an EOC event is generated
  - 3rd trigger: channel 7 is converted and an EOC event is generated
  - 4th trigger: channel 10 is converted and both EOC and EOS events are generated.
  - 5th trigger: channel 0 is converted an EOC event is generated
  - 6th trigger: channel 3 is converted and an EOC event is generated
  - ...

- DISCEN = 0, channels to be converted = 0, 3, 7, 10
  - 1st trigger: the complete sequence is converted: channel 0, then 3, 7 and 10. Each conversion generates an EOC event and the last one also generates an EOS event.
  - Any subsequent trigger events restarts the complete sequence.

Note: It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.

12.4.2 Programmable resolution (RES) - Fast conversion mode

It is possible to obtain faster conversion times ($t_{SAR}$) by reducing the ADC resolution.

The resolution can be configured to be either 12, 10, 8, or 6 bits by programming the RES[1:0] bits in the ADC_CFGR1 register. Lower resolution allows faster conversion times for applications where high data precision is not required.

Note: The RES[1:0] bit must only be changed when the ADEN bit is reset.

The result of the conversion is always 12 bits wide and any unused LSB bits are read as zeros.

Lower resolution reduces the conversion time needed for the successive approximation steps as shown in Table 38.
12.4.3 End of conversion, end of sampling phase (EOC, EOSMP flags)

The ADC indicates each end of conversion (EOC) event.

The ADC sets the EOC flag in the ADC_ISR register as soon as a new conversion data result is available in the ADC_DR register. An interrupt can be generated if the EOCIE bit is set in the ADC_IER register. The EOC flag is cleared by software either by writing 1 to it, or by reading the ADC_DR register.

The ADC also indicates the end of sampling phase by setting the EOSMP flag in the ADC_ISR register. The EOSMP flag is cleared by software by writing 1 to it. An interrupt can be generated if the EOSMPIE bit is set in the ADC_IER register.

The aim of this interrupt is to allow the processing to be synchronized with the conversions. Typically, an analog multiplexer can be accessed in hidden time during the conversion phase, so that the multiplexer is positioned when the next sampling starts.

Note: As there is only a very short time left between the end of the sampling and the end of the conversion, it is recommended to use polling or a WFE instruction rather than an interrupt and a WFI instruction.

12.4.4 End of conversion sequence (EOS flag)

The ADC notifies the application of each end of sequence (EOS) event.

The ADC sets the EOS flag in the ADC_ISR register as soon as the last data result of a conversion sequence is available in the ADC_DR register. An interrupt can be generated if the EOSIE bit is set in the ADC_IER register. The EOS flag is cleared by software by writing 1 to it.
### 12.4.5 Example timing diagrams (single/continuous modes hardware/software triggers)

#### Figure 30. Single conversions of a sequence, software trigger

<table>
<thead>
<tr>
<th>ADSTART&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>EOC</th>
<th>EOS</th>
<th>SCANDIR</th>
<th>ADC state&lt;sup&gt;(2)&lt;/sup&gt;</th>
<th>ADC_DR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RDY, CH0, CH9, CH10, CH17, RDY, CH17, CH10, CH9, CH0, RDY</td>
<td>D0, D9, D10, D17, D17, D10, D9, D0</td>
</tr>
</tbody>
</table>

1. EXTEP = 00, CONT = 0
2. CHSEL = 0x20601, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section A.7.5: Single conversion sequence - software trigger.

#### Figure 31. Continuous conversion of a sequence, software trigger

<table>
<thead>
<tr>
<th>ADSTART&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>EOC</th>
<th>EOS</th>
<th>ADSTP</th>
<th>SCANDIR</th>
<th>ADC state&lt;sup&gt;(2)&lt;/sup&gt;</th>
<th>ADC_DR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RDY, CH0, CH9, CH10, CH17, CH0, CH9, CH10, STP, RDY, CH17, CH10</td>
<td>D0, D9, D10, D17, D0, D9, D17</td>
</tr>
</tbody>
</table>

1. EXTEP = 00, CONT = 1,
2. CHSEL = 0x20601, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section A.7.6: Continuous conversion sequence - software trigger.
Figure 32. Single conversions of a sequence, hardware trigger

1. EXTSEL = TRGx (over-frequency), EXTEN = 01 (rising edge), CONT = 0
2. CHSEL = 0xF, SCANDIR = 0, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section A.7.7: Single conversion sequence - hardware trigger.

Figure 33. Continuous conversions of a sequence, hardware trigger

1. EXTSEL = TRGx, EXTEN = 10 (falling edge), CONT = 1
2. CHSEL = 0xF, SCANDIR = 0, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section A.7.8: Continuous conversion sequence - hardware trigger.
12.5 Data management

12.5.1 Data register and data alignment (ADC_DR, ALIGN)

At the end of each conversion (when an EOC event occurs), the result of the converted data is stored in the ADC_DR data register which is 16-bit wide.

The format of the ADC_DR depends on the configured data alignment and resolution. The ALIGN bit in the ADC_CFGR1 register selects the alignment of the data stored after conversion. Data can be right-aligned (ALIGN = 0) or left-aligned (ALIGN = 1) as shown in Figure 34.

**Figure 34. Data alignment and resolution**

| ALIGN | RES | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 0     | 0x0 | 0x0 |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 0x1   |     | 0x00|    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 0x2   |     |    | 0x0 |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 0x3   |     |    |    | 0x00|    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0x0 |   |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 0x1   |   |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 0x2   |   |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 0x3   |   |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |

12.5.2 ADC overrun (OVR, OVRMOD)

The overrun flag (OVR) indicates a data overrun event, when the converted data was not read in time by the CPU or the DMA, before the data from a new conversion is available.

The OVR flag is set in the ADC_ISR register if the EOC flag is still at ‘1’ at the time when a new conversion completes. An interrupt can be generated if the OVRIE bit is set in the ADC_IER register.

When an overrun condition occurs, the ADC keeps operating and can continue to convert unless the software decides to stop and reset the sequence by setting the ADSTP bit in the ADC_CR register.

The OVR flag is cleared by software by writing 1 to it.

It is possible to configure if the data is preserved or overwritten when an overrun event occurs by programming the OVRMOD bit in the ADC_CFGR1 register:

- **OVRMOD = 0**
  - An overrun event preserves the data register from being overwritten: the old data is maintained and the new conversion is discarded. If OVR remains at 1, further conversions can be performed but the resulting data is discarded.

- **OVRMOD = 1**
  - The data register is overwritten with the last conversion result and the previous unread data is lost. If OVR remains at 1, further conversions can be performed and the ADC_DR register always contains the data from the latest conversion.
12.5.3 Managing a sequence of data converted without using the DMA

If the conversions are slow enough, the conversion sequence can be handled by software. In this case the software must use the EOC flag and its associated interrupt to handle each data result. Each time a conversion is complete, the EOC bit is set in the ADC_ISR register and the ADC_DR register can be read. The OVRMOD bit in the ADC_CFGR1 register should be configured to 0 to manage overrun events as an error.

12.5.4 Managing converted data without using the DMA without overrun

It may be useful to let the ADC convert one or more channels without reading the data after each conversion. In this case, the OVRMOD bit must be configured at 1 and the OVR flag should be ignored by the software. When OVRMOD = 1, an overrun event does not prevent the ADC from continuing to convert and the ADC_DR register always contains the latest conversion data.

12.5.5 Managing converted data using the DMA

Since all converted channel values are stored in a single data register, it is efficient to use DMA when converting more than one channel. This avoids losing the conversion data results stored in the ADC_DR register.
When DMA mode is enabled (DMAEN bit set in the ADC_CFGR1 register), a DMA request is generated after the conversion of each channel. This allows the transfer of the converted data from the ADC_DR register to the destination location selected by the software.

**Note:** The DMAEN bit in the ADC_CFGR1 register must be set after the ADC calibration phase.

Despite this, if an overrun occurs (OVR = 1) because the DMA could not serve the DMA transfer request in time, the ADC stops generating DMA requests and the data corresponding to the new conversion is not transferred by the DMA. Which means that all the data transferred to the RAM can be considered as valid.

Depending on the configuration of OVRMOD bit, the data is either preserved or overwritten (refer to Section 12.5.2: ADC overrun (OVR, OVRMOD) on page 198).

The DMA transfer requests are blocked until the software clears the OVR bit.

Two different DMA modes are proposed depending on the application use and are configured with bit DMACFG in the ADC_CFGR1 register:

- **DMA one shot mode (DMACFG = 0)**
  - This mode should be selected when the DMA is programmed to transfer a fixed number of data words.
- **DMA circular mode (DMACFG = 1)**
  - This mode should be selected when programming the DMA in circular mode or double buffer mode.

**DMA one shot mode (DMACFG = 0)**

In this mode, the ADC generates a DMA transfer request each time a new conversion data word is available and stops generating DMA requests once the DMA has reached the last DMA transfer (when a transfer complete interrupt occurs, see Section 10: Direct memory access controller (DMA)) even if a conversion has been started again.

For code example refer to the Appendix section A.7.9: DMA one shot mode sequence.

When the DMA transfer is complete (all the transfers configured in the DMA controller have been done):

- The content of the ADC data register is frozen.
- Any ongoing conversion is aborted and its partial result discarded
- No new DMA request is issued to the DMA controller. This avoids generating an overrun error if there are still conversions which are started.
- The scan sequence is stopped and reset
- The DMA is stopped

**DMA circular mode (DMACFG = 1)**

In this mode, the ADC generates a DMA transfer request each time a new conversion data word is available in the data register, even if the DMA has reached the last DMA transfer. This allows the DMA to be configured in circular mode to handle a continuous analog input data stream.

For code example refer to the Appendix section A.7.10: DMA circular mode sequence.
12.6 Low-power features

12.6.1 Wait mode conversion

Wait mode conversion can be used to simplify the software as well as optimizing the performance of applications clocked at low frequency where there might be a risk of ADC overrun occurring.

When the WAIT bit is set in the ADC_CFRG1 register, a new conversion can start only if the previous data has been treated, once the ADC_DR register has been read or if the EOC bit has been cleared.

This is a way to automatically adapt the speed of the ADC to the speed of the system that reads the data.

*Note:* Any hardware triggers which occur while a conversion is ongoing or during the wait time preceding the read access are ignored.

**Figure 36. Wait mode conversion (continuous mode, software trigger)**

1. EXTEN = 00, CONT = 1
2. CHSEL = 0x3, SCANDIR = 0, WAIT = 1, AUTOFF = 0

For code example refer to the Appendix section A.7.11: Wait mode sequence.
12.6.2  Auto-off mode (AUTOFF)

The ADC has an automatic power management feature which is called auto-off mode, and is enabled by setting AUTOFF = 1 in the ADC_CFGR1 register.

When AUTOFF = 1, the ADC is always powered off when not converting and automatically wakes-up when a conversion is started (by software or hardware trigger). A startup-time is automatically inserted between the trigger event which starts the conversion and the sampling time of the ADC. The ADC is then automatically disabled once the sequence of conversions is complete.

Auto-off mode can cause a dramatic reduction in the power consumption of applications which need relatively few conversions or when conversion requests are timed far enough apart (for example with a low frequency hardware trigger) to justify the extra power and extra time used for switching the ADC on and off.

Auto-off mode can be combined with the wait mode conversion (WAIT = 1) for applications clocked at low frequency. This combination can provide significant power savings if the ADC is automatically powered-off during the wait phase and restarted as soon as the ADC_DR register is read by the application (see Figure 38: Behavior with WAIT = 1, AUTOFF = 1).

Note:  Refer to the Section Reset and clock control (RCC) for the description of how to manage the dedicated 14 MHz internal oscillator. The ADC interface can automatically switch ON/OFF the 14 MHz internal oscillator to save power.

Figure 37. Behavior with WAIT = 0, AUTOFF = 1

1. EXTSEL = TRGx, EXTEN = 01 (rising edge), CONT = x, ADSTART = 1, CHSEL = 0xF, SCANDIR = 0, WAIT = 1, AUTOFF = 1

For code example refer to the Appendix section A.7.12: Auto Off and no wait mode sequence.
12.7 Analog window watchdog

12.7.1 Description of the analog watchdog

The AWD analog watchdog is enabled by setting the AWDEN bit in the ADC_CFGR1 register. It is used to monitor that either one selected channel or all enabled channels (see Table 40: Analog watchdog channel selection) remain within a configured voltage range (window) as shown in Figure 39.

The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a lower threshold or above a higher threshold. These thresholds are programmed in HT[11:0] and LT[11:0] bit of ADC_TR register. An interrupt can be enabled by setting the AWDIE bit in the ADC_IER register.

The AWD flag is cleared by software by programming it to it.

When converting data with a resolution of less than 12-bit (according to bits RES[1:0]), the LSB of the programmed thresholds must be kept cleared because the internal comparison is always performed on the full 12-bit raw converted data (left aligned).

For code example refer to the Appendix section A.7.14: Analog watchdog.

Table 39 describes how the comparison is performed for all the possible resolutions.
Table 39. Analog watchdog comparison

<table>
<thead>
<tr>
<th>Resolution bits RES[1:0]</th>
<th>Analog watchdog comparison between:</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00: 12-bit</td>
<td>DATA[11:0] LT[11:0] and HT[11:0]</td>
<td>-</td>
</tr>
<tr>
<td>01: 10-bit</td>
<td>DATA[11:2],00 LT[11:0] and HT[11:0]</td>
<td>The user must configure LT[1:0] and HT[1:0] to “00”</td>
</tr>
</tbody>
</table>

1. The watchdog comparison is performed on the raw converted data before any alignment calculation.

Table 40 shows how to configure the AWDSGL and AWDEN bits in the ADC_CFGR1 register to enable the analog watchdog on one or more channels.

Figure 39. Analog watchdog guarded area

Table 40. Analog watchdog channel selection

<table>
<thead>
<tr>
<th>Channels guarded by the analog watchdog</th>
<th>AWDSGL bit</th>
<th>AWDEN bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>All channels</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Single(1) channel</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1. Selected by the AWDCH[4:0] bits

12.7.2 ADC_AWD1_OUT output signal generation

The analog watchdog is associated to an internal hardware signal, ADC_AWD1_OUT that is directly connected to the ETR input (external trigger) of some on-chip timers (refer to the timers section for details on how to select the ADC_AWD1_OUT signal as ETR).
ADC_AWD1_OUT is activated when the analog watchdog is enabled:

- ADC_AWD1_OUT is set when a guarded conversion is outside the programmed thresholds.
- ADC_AWD1_OUT is reset after the end of the next guarded conversion which is inside the programmed thresholds. It remains at 1 if the next guarded conversions are still outside the programmed thresholds.
- ADC_AWD1_OUT is also reset when disabling the ADC (when setting ADDIS to 1). Note that stopping conversions (ADSTP set), might clear the ADC_AWD1_OUT state.
- ADC_AWD1_OUT state does not change when the ADC converts the none-guarded channel (see Figure 40).

AWD flag is set by hardware and reset by software: AWD flag has no influence on the generation of ADC_AWD1_OUT (as an example, ADC_AWD1_OUT can toggle while AWD flag remains at 1 if the software has not cleared the flag).

The ADC_AWD1_OUT signal is generated by the ADC_CLK domain. This signal can be generated even the APB clock is stopped.

The AWD comparison is performed at the end of each ADC conversion. The ADC_AWD1_OUT rising edge and falling edge occurs two ADC_CLK clock cycles after the comparison.

As ADC_AWD1_OUT is generated by the ADC_CLK domain and AWD flag is generated by the APB clock domain, the rising edges of these signals are not synchronized.

**Figure 40. ADC_AWD1_OUT signal generation**

![ADC_AWD1_OUT signal generation](image)

- Converted channels: 1,2,3,4,5,6,7
- Guarded converted channels: 1,2,3,4,5,6,7
12.7.3 Analog watchdog threshold control

LT[11:0] and HT[11:0] can be changed during an analog-to-digital conversion (that is between the start of the conversion and the end of conversion of the ADC internal state). If LT and HT bits are programmed during the ADC guarded channel conversion, the watchdog function is masked for this conversion. This mask is cleared when starting a new conversion, and the resulting new AWD threshold is applied starting the next ADC conversion result. AWD comparison is performed at each end of conversion. If the current ADC data are out of the new threshold interval, this does not generate any interrupt or an ADC_AWD1_OUT signal. The Interrupt and the ADC_AWD1_OUT generation only occurs at the end of the ADC conversion that started after the threshold update. If ADC_AWD1_OUT is already asserted, programming the new threshold does not deassert the ADC_AWD1_OUT signal.
The temperature sensor can be used to measure the junction temperature ($T_{J}$) of the device. The temperature sensor is internally connected to the ADC $V_{IN}[16]$ input channel which is used to convert the sensor’s output voltage to a digital value. The sampling time for the temperature sensor analog pin must be greater than the minimum $T_{S\_temp}$ value specified in the datasheet. When not in use, the sensor can be put in power down mode.

The temperature sensor output voltage changes linearly with temperature, however its characteristics may vary significantly from chip to chip due to the process variations. To improve the accuracy of the temperature sensor (especially for absolute temperature measurement), calibration values are individually measured for each part by ST during production test and stored in the system memory area. Refer to the specific device datasheet for additional information.

The internal voltage reference ($V_{REFINT}$) provides a stable (bandgap) voltage output for the ADC and comparators. $V_{REFINT}$ is internally connected to the ADC $V_{IN}[17]$ input channel. The precise voltage of $V_{REFINT}$ is individually measured for each part by ST during production test and stored in the system memory area. It is accessible in read-only mode.

*Figure 44* shows the block diagram of connections between the temperature sensor, the internal voltage reference and the ADC.

The TSEN bit must be set to enable the conversion of ADC $V_{IN}[16]$ (temperature sensor) and the VREFEN bit must be set to enable the conversion of ADC $V_{IN}[17]$ ($V_{REFINT}$).
Main features

- Linearity: ±2 °C max., precision depending on calibration

Figure 44. Temperature sensor and VREFINT channel block diagram

Reading the temperature

1. Select the ADC V[16] input channel.
2. Select an appropriate sampling time specified in the device datasheet (T_{temp}).
3. Set the TSEN bit in the ADC_CCR register to wake up the temperature sensor from power down mode and wait for its stabilization time (t_{START}).
   For code example refer to the Appendix section A.7.15: Temperature configuration.
4. Start the ADC conversion by setting the ADSTART bit in the ADC_CR register (or by external trigger).
5. Read the resulting V_{SENSE} data in the ADC_DR register.
6. Calculate the temperature using the following formula

   \[ \text{Temperature (in °C)} = \frac{\text{Sense\_DATA} - \text{TS\_CAL1}}{\text{Avg\_Slope\_Code}} + \text{TS\_CAL1\_TEMP} \]

   \[ \text{Avg\_Slope\_Code} = \frac{\text{Avg\_Slope} \times 4096}{3300} \]

   \[ \text{Sense\_DATA} = \frac{\text{TS\_DATA} \times V_{DDA}}{3.3} \]
Where:

- TS_CAL1 is the temperature sensor calibration value acquired at TS_CAL1_TEMP (refer to the datasheet for TS_CAL1 value)
- TS_DATA is the actual temperature sensor output value converted by ADC
  Refer to the specific device datasheet for more information about TS_CAL1 calibration point.
- Avg_Slope is the coefficient of the temperature sensor output voltage expressed in mV/°C (refer to the datasheet for Avg_Slope value).

For code example refer to the A.7.16: Temperature computation.

Note: The sensor has a startup time after waking from power down mode before it can output VSENSE at the correct level. The ADC also has a startup time after power-on, so to minimize the delay, the ADEN and TSEN bits should be set at the same time.

Calculating the actual VDDA voltage using the internal reference voltage

The VDDA power supply voltage applied to the device may be subject to variation or not precisely known. The embedded internal voltage reference (VREFINT) and its calibration data, acquired by the ADC during the manufacturing process at VDDA_Charac, can be used to evaluate the actual VDDA voltage level.

The following formula gives the actual VDDA voltage supplying the device:

\[ V_{DDA} = V_{DDA_{Charac}} \times \frac{V_{REFINT_{CAL}}}{V_{REFINT_{DATA}}} \]

Where:

- \( V_{DDA_{Charac}} \) is the value of VDDA voltage characterized at VREFINT during the manufacturing process. It is specified in the device datasheet.
- \( V_{REFINT_{CAL}} \) is the VREFINT calibration value
- \( V_{REFINT_{DATA}} \) is the actual VREFINT output value converted by ADC

Converting a supply-relative ADC measurement to an absolute voltage value

The ADC is designed to deliver a digital value corresponding to the ratio between the analog power supply and the voltage applied on the converted channel. For most application use cases, it is necessary to convert this ratio into a voltage independent of VDDA. For applications where VDDA is known and ADC converted values are right-aligned you can use the following formula to get this absolute value:

\[ V_{CHANNEL} = \frac{V_{DDA} \times ADC_{DATA}}{NUM\_CODES} \]

For applications where VDDA value is not known, you must use the internal voltage reference and VDDA can be replaced by the expression provided in Section: Calculating the actual VDDA voltage using the internal reference voltage, resulting in the following formula:

\[ V_{CHANNEL} = \frac{V_{DDA_{Charac}} \times V_{REFINT_{CAL}} \times ADC_{DATA}}{V_{REFINT_{DATA}} \times NUM\_CODES} \]
Where:

- $V_{DDA_{Charac}}$ is the value of $V_{DDA}$ voltage characterized at $V_{REFINT}$ during the manufacturing process. It is specified in the device datasheet.
- $V_{REFINT\_CAL}$ is the $V_{REFINT}$ calibration value
- $ADC\_DATA_x$ is the value measured by the ADC on channel $x$ (right-aligned)
- $V_{REFINT\_DATA}$ is the actual $V_{REFINT}$ output value converted by the ADC
- $NUM\_CODES$ is the number of ADC output codes. For example with 12-bit resolution, it is $2^{12} = 4096$ or with 8-bit resolution, $2^8 = 256$.

**Note:** If ADC measurements are done using an output format other than 12 bit right-aligned, all the parameters must first be converted to a compatible format before the calculation is done.

### 12.9 ADC interrupts

An interrupt can be generated by any of the following events:

- ADC power-up, when the ADC is ready (ADRDY flag)
- End of any conversion (EOC flag)
- End of a sequence of conversions (EOS flag)
- When an analog watchdog detection occurs (AWD flag)
- When the end of sampling phase occurs (EOSMP flag)
- When a data overrun occurs (OVR flag)

Separate interrupt enable bits are available for flexibility.

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC ready</td>
<td>ADRDY</td>
<td>ADRDYIE</td>
</tr>
<tr>
<td>End of conversion</td>
<td>EOC</td>
<td>EOCIE</td>
</tr>
<tr>
<td>End of sequence of conversions</td>
<td>EOS</td>
<td>EOSIE</td>
</tr>
<tr>
<td>Analog watchdog status bit is set</td>
<td>AWD</td>
<td>AWDIE</td>
</tr>
<tr>
<td>End of sampling phase</td>
<td>EOSMP</td>
<td>EOSMPIE</td>
</tr>
<tr>
<td>Overrun</td>
<td>OVR</td>
<td>OVRIE</td>
</tr>
</tbody>
</table>
12.10 ADC registers

Refer to Section 1.2 for a list of abbreviations used in register descriptions.

12.10.1 ADC interrupt and status register (ADC_ISR)

Address offset: 0x00
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rc_w1</td>
<td>rc_w1</td>
<td>rc_w1</td>
<td>rc_w1</td>
<td>rc_w1</td>
<td>rc_w1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:13 Reserved, must be kept at reset value.

Bits 12:10 Reserved, must be kept at reset value.

Bits 9:8 Reserved, must be kept at reset value.

Bit 7 AWD: Analog watchdog flag
This bit is set by hardware when the converted voltage crosses the values programmed in ADC_TR register. It is cleared by software by programming it to 1.
0: No analog watchdog event occurred (or the flag event was already acknowledged and cleared by software)
1: Analog watchdog event occurred

Bits 6:5 Reserved, must be kept at reset value.

Bit 4 OVR: ADC overrun
This bit is set by hardware when an overrun occurs, meaning that a new conversion has complete while the EOC flag was already set. It is cleared by software writing 1 to it.
0: No overrun occurred (or the flag event was already acknowledged and cleared by software)
1: Overrun has occurred

Bit 3 EOS: End of sequence flag
This bit is set by hardware at the end of the conversion of a sequence of channels selected by the CHSEL bits. It is cleared by software writing 1 to it.
0: Conversion sequence not complete (or the flag event was already acknowledged and cleared by software)
1: Conversion sequence complete
Bit 2  **EOC**: End of conversion flag  
This bit is set by hardware at the end of each conversion of a channel when a new data result is available in the ADC_DR register. It is cleared by software writing 1 to it or by reading the ADC_DR register.

- 0: Channel conversion not complete (or the flag event was already acknowledged and cleared by software)
- 1: Channel conversion complete

Bit 1  **EOSMP**: End of sampling flag  
This bit is set by hardware during the conversion, at the end of the sampling phase. It is cleared by software by programming it to ‘1’.

- 0: Not at the end of the sampling phase (or the flag event was already acknowledged and cleared by software)
- 1: End of sampling phase reached

Bit 0  **ADRDY**: ADC ready  
This bit is set by hardware after the ADC has been enabled (ADEN = 1) and when the ADC reaches a state where it is ready to accept conversion requests.

- 0: ADC not yet ready to start conversion (or the flag event was already acknowledged and cleared by software)
- 1: ADC is ready to start conversion

**Note:** In auto-off mode (AUTOFF = 1) the power-on/off phases are performed automatically, by hardware and the ADRDY flag is not set.

### 12.10.2 ADC interrupt enable register (ADC_IER)

#### Address offset: 0x04

Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>15</td>
<td>AWDIE</td>
<td>Analog watchdog interrupt enable</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>11</td>
<td>OVRIE</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>10</td>
<td>EOSIE</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>9</td>
<td>EOCIE</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>8</td>
<td>EOSMPIE</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>7</td>
<td>ADRDYIE</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>

Bits 31:13 Reserved, must be kept at reset value.

Bits 12:10 Reserved, must be kept at reset value.

Bits 9:8 Reserved, must be kept at reset value.

Bit 7  **AWDIE**: Analog watchdog interrupt enable  
This bit is set and cleared by software to enable/disable the analog watchdog interrupt.

- 0: Analog watchdog interrupt disabled
- 1: Analog watchdog interrupt enabled

**Note:** The Software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bits 6:5 Reserved, must be kept at reset value.
Bit 4 **OVRIE**: Overrun interrupt enable
This bit is set and cleared by software to enable/disable the overrun interrupt.
0: Overrun interrupt disabled
1: Overrun interrupt enabled. An interrupt is generated when the OVR bit is set.
*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bit 3 **EOSIE**: End of conversion sequence interrupt enable
This bit is set and cleared by software to enable/disable the end of sequence of conversions interrupt.
0: EOS interrupt disabled
1: EOS interrupt enabled. An interrupt is generated when the EOS bit is set.
*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bit 2 **EOCIE**: End of conversion interrupt enable
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.
*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bit 1 **EOSMPIE**: End of sampling flag interrupt enable
This bit is set and cleared by software to enable/disable the end of the sampling phase interrupt.
0: EOSMP interrupt disabled
1: EOSMP interrupt enabled. An interrupt is generated when the EOSMP bit is set.
*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bit 0 **ADRDYIE**: ADC ready interrupt enable
This bit is set and cleared by software to enable/disable the ADC Ready interrupt.
0: ADRDY interrupt disabled
1: ADRDY interrupt enabled. An interrupt is generated when the ADRDY bit is set.
*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*
12.10.3 ADC control register (ADC_CR)

Address offset: 0x08
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>ADCAL: ADC calibration</th>
</tr>
</thead>
<tbody>
<tr>
<td>This bit is set by software to start the calibration of the ADC.</td>
<td></td>
</tr>
<tr>
<td>It is cleared by hardware after calibration is complete.</td>
<td></td>
</tr>
<tr>
<td>0: Calibration complete</td>
<td></td>
</tr>
<tr>
<td>1: Write 1 to calibrate the ADC. Read at 1 means that a calibration is in progress.</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The software is allowed to set ADCAL only when the ADC is disabled (ADCAL = 0, ADSTART = 0, ADSTP = 0, ADDIS = 0, AUTOFF = 0, and ADEN = 0).*

Bits 30:28 Reserved, must be kept at reset value.

Bits 27:5 Reserved, must be kept at reset value.

Bit 4 ADSTP: ADC stop conversion command

This bit is set by software to stop and discard an ongoing conversion (ADSTP Command).
It is cleared by hardware when the conversion is effectively discarded and the ADC is ready to accept a new start conversion command.

0: No ADC stop conversion command ongoing
1: Write 1 to stop the ADC. Read 1 means that an ADSTP command is in progress.

*Note: Setting ADSTP to ‘1’ is only effective when ADSTART = 1 and ADDIS = 0 (ADC is enabled and may be converting and there is no pending request to disable the ADC)*

Bit 3 Reserved, must be kept at reset value.
Bit 2  **ADSTART**: ADC start conversion command

This bit is set by software to start ADC conversion. Depending on the EXTEN [1:0] configuration bits, a conversion either starts immediately (software trigger configuration) or once a hardware trigger event occurs (hardware trigger configuration).

It is cleared by hardware:
- In single conversion mode (CONT = 0, DISCEN = 0), when software trigger is selected (EXTEN = 00): at the assertion of the end of Conversion Sequence (EOS) flag.
- In discontinuous conversion mode (CONT = 0, DISCEN = 1), when the software trigger is selected (EXTEN = 00): at the assertion of the end of Conversion (EOC) flag.
- In all other cases: after the execution of the ADSTP command, at the same time as the ADSTP bit is cleared by hardware.

0: No ADC conversion is ongoing.
1: Write 1 to start the ADC. Read 1 means that the ADC is operating and may be converting.

*Note:* The software is allowed to set ADSTART only when ADEN = 1 and ADDIS = 0 (ADC is enabled and there is no pending request to disable the ADC).

Bit 1  **ADDIS**: ADC disable command

This bit is set by software to disable the ADC (ADDIS command) and put it into power-down state (OFF state).

It is cleared by hardware once the ADC is effectively disabled (ADEN is also cleared by hardware at this time).

0: No ADDIS command ongoing
1: Write 1 to disable the ADC. Read 1 means that an ADDIS command is in progress.

*Note:* Setting ADDIS to ‘1’ is only effective when ADEN = 1 and ADSTART = 0 (which ensures that no conversion is ongoing)

Bit 0  **ADEN**: ADC enable command

This bit is set by software to enable the ADC. The ADC is effectively ready to operate once the ADRDY flag has been set.

It is cleared by hardware when the ADC is disabled, after the execution of the ADDIS command.

0: ADC is disabled (OFF state)
1: Write 1 to enable the ADC.

*Note:* The software is allowed to set ADEN only when all bits of ADC_CR registers are 0 (ADCAL = 0, ADSTP = 0, ADSTART = 0, ADDIS = 0 and ADEN = 0)
### 12.10.4 ADC configuration register 1 (ADC_CFGR1)

Address offset: 0x0C  
Reset value: 0x0000 0000

| Bit 31 | Reserved, must be kept at reset value. |
| Bit 30-26 | **AWDCH[4:0]**: Analog watchdog channel selection |
| | These bits are set and cleared by software. They select the input channel to be guarded by the analog watchdog. |
| | 00000: ADC analog input Channel 0 monitored by AWD |
| | 00001: ADC analog input Channel 1 monitored by AWD |
| | ..... |
| | 10001: ADC analog input Channel 17 monitored by AWD |
| | Others: Reserved |
| | **Note**: The channel selected by the AWDCH[4:0] bits must be also set into the CHSELR register. |
| | **The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).** |
| Bit 25-24 | Reserved, must be kept at reset value. |
| Bit 23 | **AWDEN**: Analog watchdog enable |
| | This bit is set and cleared by software. |
| | 0: Analog watchdog disabled |
| | 1: Analog watchdog enabled |
| | **Note**: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing). |
| Bit 22 | **AWDSGL**: Enable the watchdog on a single channel or on all channels |
| | This bit is set and cleared by software to enable the analog watchdog on the channel identified by the AWDCH[4:0] bits or on all the channels |
| | 0: Analog watchdog enabled on all channels |
| | 1: Analog watchdog enabled on a single channel |
| | **Note**: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing). |
| Bit 21-17 | Reserved, must be kept at reset value. |
Bit 16 **DISCEN**: Discontinuous mode
This bit is set and cleared by software to enable/disable discontinuous mode.
0: Discontinuous mode disabled
1: Discontinuous mode enabled

*Note*: It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.
The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 15 **AUTOFF**: Auto-off mode
This bit is set and cleared by software to enable/disable auto-off mode.
0: Auto-off mode disabled
1: Auto-off mode enabled

*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 14 **WAIT**: Wait conversion mode
This bit is set and cleared by software to enable/disable wait conversion mode.
0: Wait conversion mode off
1: Wait conversion mode on

*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 13 **CONT**: Single / continuous conversion mode
This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.
0: Single conversion mode
1: Continuous conversion mode

*Note*: It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.
The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 12 **OVRMOD**: Overrun management mode
This bit is set and cleared by software and configure the way data overruns are managed.
0: ADC_DR register is preserved with the old data when an overrun is detected.
1: ADC_DR register is overwritten with the last conversion result when an overrun is detected.

*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bits 11:10 **EXTEN[1:0]**: External trigger enable and polarity selection
These bits are set and cleared by software to select the external trigger polarity and enable the trigger.
00: Hardware trigger detection disabled (conversions can be started by software)
01: Hardware trigger detection on the rising edge
10: Hardware trigger detection on the falling edge
11: Hardware trigger detection on both the rising and falling edges

*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).
Bit 9  Reserved, must be kept at reset value.

Bits 8:6  **EXTSEL[2:0]**: External trigger selection

These bits select the external event used to trigger the start of conversion (refer to *Table 35: External triggers* for details):

- 000: TRG0
- 001: TRG1
- 010: TRG2
- 011: TRG3
- 100: TRG4
- 101: TRG5
- 110: TRG6
- 111: TRG7

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing)*.

Bit 5  **ALIGN**: Data alignment

This bit is set and cleared by software to select right or left alignment. Refer to *Figure 34: Data alignment and resolution on page 198*

- 0: Right alignment
- 1: Left alignment

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing)*.

Bits 4:3  **RES[1:0]**: Data resolution

These bits are written by software to select the resolution of the conversion.

- 00: 12 bits
- 01: 10 bits
- 10: 8 bits
- 11: 6 bits
Bit 2  **SCANDIR**: Scan sequence direction
This bit is set and cleared by software to select the direction in which the channels are scanned in the sequence.
0: Upward scan (from CHSEL0 to CHSEL17)
1: Backward scan (from CHSEL17 to CHSEL0)
*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 1  **DMACFG**: Direct memory access configuration
This bit is set and cleared by software to select between two DMA modes of operation and is effective only when DMAEN = 1.
0: DMA one shot mode selected
1: DMA circular mode selected
For more details, refer to [Section 12.5.5: Managing converted data using the DMA on page 199.](#)
*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 0  **DMAEN**: Direct memory access enable
This bit is set and cleared by software to enable the generation of DMA requests. This allows the DMA controller to be used to manage automatically the converted data. For more details, refer to [Section 12.5.5: Managing converted data using the DMA on page 199.](#)
0: DMA disabled
1: DMA enabled
*Note*: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).
12.10.5  ADC configuration register 2 (ADC_CFRG2)

Address offset: 0x10
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bits 31:30</th>
<th>CKMODE[1:0]: ADC clock mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>ADCCLK (Asynchronous clock mode), generated at product level (refer to RCC section)</td>
</tr>
<tr>
<td>01</td>
<td>PCLK/2 (Synchronous clock mode)</td>
</tr>
<tr>
<td>10</td>
<td>PCLK/4 (Synchronous clock mode)</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

In all synchronous clock modes, there is no jitter in the delay from a timer trigger to the start of a conversion.

*Note: The software is allowed to write these bits only when the ADC is disabled (ADCAL = 0, ADSTART = 0, ADSTP = 0, ADDIS = 0 and ADEN = 0).*

Bits 29:10  Reserved, must be kept at reset value.

Bits 9:0    Reserved, must be kept at reset value.

12.10.6  ADC sampling time register (ADC_SMPR)

Address offset: 0x14
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bits 31:30</th>
<th>SMP[2:0]</th>
</tr>
</thead>
</table>

Note: The software is allowed to write these bits only when the ADC is disabled (ADCAL = 0, ADSTART = 0, ADSTP = 0, ADDIS = 0 and ADEN = 0).
12.10.7 ADC watchdog threshold register (ADC_TR)

Address offset: 0x20
Reset value: 0x0FFF 0000

| Bits 31:28 | Reserved, must be kept at reset value. |
| Bits 27:16 | **HT[11:0]**: Analog watchdog higher threshold |
| These bits are written by software to define the higher threshold for the analog watchdog. Refer to Section 12.7: Analog window watchdog on page 203 |
| Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing). |
| Bits 15:12 | Reserved, must be kept at reset value. |
| Bits 11:0 | **LT[11:0]**: Analog watchdog lower threshold |
| These bits are written by software to define the lower threshold for the analog watchdog. Refer to Section 12.7: Analog window watchdog on page 203. |
| Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing). |

12.10.8 ADC channel selection register (ADC_CHSELR)

Address offset: 0x28
Reset value: 0x0000 0000

Bits 31:3 Reserved, must be kept at reset value.

Bits 2:0 **SMP[2:0]**: Sampling time selection
These bits are written by software to select the sampling time that applies to all channels.
- 000: 1.5 ADC clock cycles
- 001: 7.5 ADC clock cycles
- 010: 13.5 ADC clock cycles
- 011: 28.5 ADC clock cycles
- 100: 41.5 ADC clock cycles
- 101: 55.5 ADC clock cycles
- 110: 71.5 ADC clock cycles
- 111: 239.5 ADC clock cycles

Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).
### ADC data register (ADC_DR)

**Address offset:** 0x40  
**Reset value:** 0x0000 0000

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
<th>Setting</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:16</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:0</td>
<td><strong>DATA[15:0]: Converted data</strong></td>
<td></td>
<td>These bits are read-only. They contain the conversion result from the last converted channel. The data are left- or right-aligned as shown in Figure 34: Data alignment and resolution on page 198. Just after a calibration is complete, DATA[6:0] contains the calibration factor.</td>
</tr>
</tbody>
</table>

#### CHSELx: Channel-x selection

These bits are written by software and define which channels are part of the sequence of channels to be converted.

- 0: Input Channel-x is not selected for conversion
- 1: Input Channel-x is selected for conversion

**Note:** The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).
12.10.10 ADC common configuration register (ADC_CCR)

Address offset: 0x308
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>Bits 31:25 Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bit 24 Reserved, must be kept at reset value.

Bit 23 **TSEN**: Temperature sensor enable
This bit is set and cleared by software to enable/disable the temperature sensor.
- 0: Temperature sensor disabled
- 1: Temperature sensor enabled

*Note*: Software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).

Bit 22 **VREFEN**: VREFINT enable
This bit is set and cleared by software to enable/disable the VREFINT.
- 0: VREFINT disabled
- 1: VREFINT enabled

*Note*: Software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).

Bits 21:0 Reserved, must be kept at reset value.

12.11 ADC register map

The following table summarizes the ADC registers.

<table>
<thead>
<tr>
<th>Table 42. ADC register map and reset values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Refer to Section 2.2 for the register boundary addresses.
13 Advanced-control timers (TIM1)

13.1 TIM1 introduction

The advanced-control timers (TIM1) 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) and general-purpose (TIMx) timers are completely independent, and do not share any resources. They can be synchronized together as described in Section 13.3.20.

13.2 TIM1 main features

TIM1 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 modes)
  - 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 circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management
Figure 45. Advanced-control timer block diagram

Notes:
- Default registers initialized: 2 capture registers on an event according to control bit
- Interrupt & DMA output

CK_TIM1 from RCC
Internal Clock (CK_INT)
ETRF
Trigger Controller
TRGO
to other timers
Slave Mode Controller
Stop, Enable, Up/Down, Count
Prescaler
AutoReload Register
COUNTER
Capture/Compare 1 Register
Capture/Compare 2 Register
Capture/Compare 3 Register
Capture/Compare 4 Register
REP Register
Repetition counter
U
Trigger Controller
Controller
Trigger
ETR
Encoder
Interface
Input Filter & Edge detector
Input Filter & Edge detector
Input Filter & Edge detector
Input Filter & Edge detector
Input Filter & Edge detector
Internal Clock (CK_INT)
Clock failure event from clock controller
CSS (Clock Security system)
13.3 TIM1 functional description

13.3.1 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 47* and *Figure 48* give some examples of the counter behavior when the prescaler ratio is changed on the fly:
Figure 46. Counter timing diagram with prescaler division change from 1 to 2

Figure 47. 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.

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 48. Counter timing diagram, internal clock divided by 1

Figure 49. Counter timing diagram, internal clock divided by 2
Figure 50. Counter timing diagram, internal clock divided by 4

Figure 51. Counter timing diagram, internal clock divided by N
Figure 52. Counter timing diagram, update event when ARPE=0
(TIMx_ARR not preloaded)

Figure 53. 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 54. Counter timing diagram, internal clock divided by 1**

```plaintext
<table>
<thead>
<tr>
<th>CK_PSC</th>
<th>CNT_EN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```plaintext
<table>
<thead>
<tr>
<th>Timer clock = CK_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter register</td>
</tr>
<tr>
<td>05 04 03 02 01 00 35 34 33 32 31 30 2F 04 03 02 01 00</td>
</tr>
</tbody>
</table>
```

```plaintext
<table>
<thead>
<tr>
<th>Counter underflow (cnt_udf)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```plaintext
<table>
<thead>
<tr>
<th>Update event (UEV)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```plaintext
<table>
<thead>
<tr>
<th>Update interrupt flag (UIF)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```
Figure 55. Counter timing diagram, internal clock divided by 2

![Counter timing diagram, internal clock divided by 2](image1)

Figure 56. Counter timing diagram, internal clock divided by 4

![Counter timing diagram, internal clock divided by 4](image2)

Figure 57. Counter timing diagram, internal clock divided by N

![Counter timing diagram, internal clock divided by N](image3)
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.

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are not equal to '00'. The Output compare interrupt flag of channels configured in output is set when: the counter counts down (Center aligned mode 1, CMS = "01"), the counter counts up (Center aligned mode 2, CMS = "10") the counter counts up and down (Center aligned mode 3, CMS = "11").

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 59. 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: TIM1 registers on page 272).
1. Center-aligned mode 2 or 3 is used with an UIF on overflow.
Figure 62. Counter timing diagram, internal clock divided by N

CK_PSC

Timer clock = CK_CNT

Counter register

Counter underflow

Update event (UEV)

Update interrupt flag (UIF)

Figure 63. Counter timing diagram, update event with ARPE=1 (counter underflow)

CK_PSC

CEN

Timer clock = CK_CNT

Counter register

Counter underflow

Update event (UEV)

Update interrupt flag (UIF)

Auto-reload preload register

Write a new value in TIMx_ARR

Auto-reload active register
13.3.3 Repetition counter

Section 13.3.1: 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 2xTck, 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 65). 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.
In center-aligned mode, for odd values of RCR, the update event occurs either on the overflow or on the underflow depending on when the RCR register was written and when the counter was started. If the RCR was written before starting the counter, the UEV occurs on the overflow. If the RCR was written after starting the counter, the UEV occurs on the underflow. For example for RCR = 3, the UEV is generated on each 4th overflow or underflow event depending on when RCR was written.

**Figure 65. Update rate examples depending on mode and TIMx_RCR register settings**

- **Counter-aligned mode**
  - TIMx_RCR = 0
  - TIMx_RCR = 1
  - TIMx_RCR = 2
  - TIMx_RCR = 3
  - TIMx_RCR = 3 and re-synchronization

- **Edge-aligned mode**
  - Upcounting
  - Downcounting

**UEV: Update event** Preload registers transferred to active registers and update interrupt generated
13.3.4 Clock sources

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, Timer 1 can be configured to act as a prescaler for Timer 2. Refer to Using one timer as prescaler for another on page 337 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 66* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

*Figure 66. 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 67. 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 it does not need to be configured.

For code examples refer to the Appendix section A.8.1: Upcounter on TI2 rising edge.

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.
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 69 gives an overview of the 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.

For code example refer to the Appendix section A.8.2: Up counter on 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 70. Control circuit in external clock mode 2**

---

### 13.3.5 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 71 to Figure 74* 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 71. 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 72. Capture/compare channel 1 main circuit
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.
13.3.6 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 it is written with '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 appropriate input filter duration in relation with the signal connected 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 most 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 fDTS 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 and CC1NP bits 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.

For code example refer to the Appendix section A.8.3: Input capture configuration.

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.

For code example refer to the Appendix section A.8.4: Input capture data management.

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.
13.3.7 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, one 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 and CC1NP bits 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' and the CC2NP bit to '0' (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.

For code example refer to the Appendix section A.8.5: PWM input configuration.

Figure 75. PWM input mode timing
13.3.8 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, one just needs 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.

13.3.9 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 CCxE 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.

For code example refer to the Appendix section A.8.7: Output compare configuration.

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 76.

**Figure 76. Output compare mode, toggle on OC1**

![Diagram of output compare mode](image)

### 13.3.10 PWM mode

Pulse Width Modulation mode allows 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. The corresponding preload register must be enabled 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, all registers must be initialized by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCRER 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_CCRER and TIMx_BDTR registers). Refer to the TIMx_CCRER 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 229*.

  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 77* shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.
For code example refer to the Appendix section **A.8.8: Edge-aligned PWM configuration example**.

- **Downcounting configuration**
  
  Downcounting is active when DIR bit in TIMx_CR1 register is high. Refer to the *Downcounting mode on page 233*

  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 235.

*Figure 78* 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.

For code example refer to the Appendix section A.8.9: Center-aligned PWM configuration example.

*Figure 78. 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 a value greater than the auto-reload value is written in the counter (TIMx_CNT>TIMx_ARR). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if 0 or the TIMx_ARR value is written 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.

13.3.11 Complementary outputs and dead-time insertion

The advanced-control timers (TIM1) 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 it has to be adjusted depending on the devices that are connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

The polarity of the outputs (main output OCx or complementary OCxN) can be selected 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 45: Output control bits for complementary OCx and OCxN channels with break feature on page 290 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)
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 13.4.18: TIM1 break and dead-time register (TIM1_BDTR) on page 295 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 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.

13.3.12 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 45: Output control bits for complementary OCx and OCxN channels with break feature on page 290 for more details.

The source for break (BRK) channel can be an external source connected to the BKIN pin or one of the following internal sources:
- the core LOCKUP output
- the PVD output
- the SRAM parity error signal
- a clock failure event generated by the CSS detector
- the output from a comparator

When exiting from reset, the break circuit is disabled and the MOE bit is low. The break function can be enabled 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. When the BKE and BKP bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

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 MOE is set to 1 whereas it was low, a delay must be inserted (dummy instruction) before reading it correctly. This is because the write acts on the asynchronous signal whereas the read reflects 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.

- 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 it is written with 1 again. In this case, it can be used for security and the break input can be connected 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 to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). The protection can be selected among 3 levels with the LOCK bits in the TIMx_BDTR register. Refer to Section 13.4.18: TIM1 break and dead-time register (TIM1_BDTR) on page 295. The LOCK bits can be written only once after an MCU reset.

The Figure 82 shows an example of behavior of the outputs in response to a break.
Figure 82. Output behavior in response to a break

<table>
<thead>
<tr>
<th>OCxREF</th>
<th>OCx</th>
<th>OCx</th>
<th>OCx</th>
<th>OCx</th>
<th>OCx</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCx</td>
<td>OCx</td>
<td>OCx</td>
<td>OCx</td>
<td>OCx</td>
<td>OCx</td>
</tr>
<tr>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
</tr>
<tr>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
</tr>
<tr>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
<td>OCxN</td>
</tr>
</tbody>
</table>

**OCx**:
- OCx not implemented, CCxP=0, OISx=1
- OCx not implemented, CCxP=0, OISx=0
- OCx not implemented, CCxP=1, OISx=1
- OCx not implemented, CCxP=1, OISx=0
- OCxN

**OCx**:
- CCxE=1, CCxP=0, OISx=0, CCxNE=1, CCxNP=0, OISxN=1
- CCxE=1, CCxP=0, OISx=1, CCxNE=1, CCxNP=1, OISxN=1
- CCxE=1, CCxP=0, OISx=0, CCxNE=0, CCxNP=0, OISxN=0 or OISxN=1

**OCx**
- CCxE=1, CCxP=0, CCxNE=1, CCxNP=0, OISx=OISxN=0
13.3.13 Clearing the OCxREF signal on an external event

The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT (OCxCE enable bit in the corresponding TIMx_CCMRx register set to 1). OCxREF remains low until the next update event (UEV) occurs. This function can only be used in Output compare and PWM modes. It does not work in Forced mode.

OCREF_CLR_INPUT can be selected between the OCREF_CLR input and ETRF (ETR after the filter) by configuring the OCCS bit in the TIMx_SMCR register.

When ETRF is chosen, ETR must be configured as follows:

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.

For code example refer to the Appendix section A.8.10: ETR configuration to clear OCxREF.

Figure 83 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 83. Clearing TIMx OCxREF

Note: In case of a PWM with a 100% duty cycle (if CCRx>ARR), then OCxREF is enabled again at the next counter overflow.
13.3.14 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 one 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 84 describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

Figure 84. 6-step generation, COM example (OSSR=1)
13.3.15 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. One-pulse mode is selected 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 85. Example of one pulse mode](image)

For example one 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’ and CC2NP=’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 $t_{\text{DELAY}}$ is defined by the value written in the TIMx_CCR1 register.
- The $t_{\text{PULSE}}$ is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1 + 1).
- Let’s say one wants 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 PWM mode 2 must be enabled by writing OC1M=111 in the TIMx_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE=’1’ in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case one has 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.

For code example refer to the Appendix section A.8.16: One-Pulse mode.

Since only 1 pulse (Single mode) is needed, a 1 must be written 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). When OPM bit in the TIMx_CR1 register is set to ‘0’, so the Repetitive Mode is selected.

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 $t_{\text{DELAY}}$ min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set 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.

For code example refer to the part of code, conditioned by PULSE_WITHOUT_DELAY > 0 in the Appendix section A.8.16: One-Pulse mode.

13.3.16 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, the input filter can be programmed as well. CC1NP and CC2NP must be kept low.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to Table 43. 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 the TIMx_ARR must be configured 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 do not switch at the same time.

<table>
<thead>
<tr>
<th>Active edge</th>
<th>TI1FP1 signal</th>
<th>TI2FP2 signal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rising</td>
<td>Falling</td>
</tr>
<tr>
<td>Counting on TI1 only</td>
<td>High</td>
<td>Down</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Up</td>
</tr>
<tr>
<td>Counting on TI2 only</td>
<td>High</td>
<td>No Count</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>No Count</td>
</tr>
<tr>
<td>Counting on TI1 and TI2</td>
<td>High</td>
<td>Down</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Up</td>
</tr>
</tbody>
</table>

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.

*Figure 86* 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).

For code example refer to the Appendix section *A.8.11: Encoder interface.*
The timer, when configured in Encoder Interface mode provides information on the sensor’s current position. Dynamic information can be obtained (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. This can be done 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.
13.3.17 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 13.3.18 below.

13.3.18 Interfacing with Hall sensors

This is done using the advanced-control timers (TIM1) to generate PWM signals to drive the motor and another timer (TIM3) referred to as “interfacing timer” in Figure 88. 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 71: Capture/compare channel (example: channel 1 input stage) on page 245). 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) (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) through the TRGO output.

Example: one wants to change the PWM configuration of the 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 XORed 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 channel 1 in capture mode (TRC selected): write the CC1S bits in the TIMx_CCMR1 register to ‘01’. The digital filter can also be programmed if needed,
- Program 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’,
Figure 88 describes this example.

Figure 88. Example of hall sensor interface

Write CCxPE, CCxNE and OCxM for next step
### 13.3.19 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 do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx_CCMR1 register. Write CC1P=0 and CC1NP='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.

For code example refer to the Appendix section A.8.12: Reset mode.

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 89. Control circuit in reset mode](image-url)
**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 do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx_CCMR1 register. Write CC1P=1 and CC1NP=’0’ 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).

For code example refer to the Appendix section A.8.13: Gated mode.

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 90. Control circuit in gated mode**

![Control circuit in gated mode](image)
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 do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC2S bits are configured to select the input capture source only, CC2S=01 in TIMx_CCMR1 register. Write CC2P=1 and CC2NP=0 in TIMx_CCR1 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.

For code example refer to the Appendix section A.8.14: Trigger mode.

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 91. 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=01 in TIMx_CCMR1 register to select only the input capture source
   – CC1P=0 and CC1NP=’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.

For code example refer to the Appendix section A.8.15: External clock mode 2 + trigger mode.

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 92. Control circuit in external clock mode 2 + trigger mode

13.3.20 Timer synchronization

The TIM timers are linked together internally for timer synchronization or chaining. Refer to Section 14.3.15: Timer synchronization on page 336 for details.

13.3.21 Debug mode

When the microcontroller enters debug mode (Cortex™-M0 core halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module.
13.4 **TIM1 registers**

Refer to [Section 1.2 on page 33](#) for a list of abbreviations used in register descriptions.

13.4.1 **TIM1 control register 1 (TIM1_CR1)**

Address offset: 0x00

Reset value: 0x0000

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:10 Reserved, must be kept at reset value.

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 = 2t_{CK\_INT}
- 10: t_DTS = 4t_{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

---

**13.4.2 TIM1 control register 2 (TIM1_CR2)**

Address offset: 0x04

Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>14</td>
<td><strong>OIS4</strong>: Output Idle state 4 (OC4 output)</td>
</tr>
<tr>
<td>13</td>
<td>refer to OIS1 bit</td>
</tr>
<tr>
<td>12</td>
<td><strong>OIS3N</strong>: Output Idle state 3 (OC3N output)</td>
</tr>
<tr>
<td>11</td>
<td>refer to OIS1N bit</td>
</tr>
<tr>
<td>10</td>
<td><strong>OIS3</strong>: Output Idle state 3 (OC3 output)</td>
</tr>
<tr>
<td>9</td>
<td>refer to OIS1 bit</td>
</tr>
<tr>
<td>8</td>
<td><strong>OIS2N</strong>: Output Idle state 2 (OC2N output)</td>
</tr>
<tr>
<td>7</td>
<td>refer to OIS1N bit</td>
</tr>
<tr>
<td>6</td>
<td><strong>OIS2</strong>: Output Idle state 2 (OC2 output)</td>
</tr>
<tr>
<td>5</td>
<td>refer to OIS1 bit</td>
</tr>
</tbody>
</table>
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 cannot be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR 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 cannot be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR 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[2:0]**: 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 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 sends 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.

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 COMG bit only
1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

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 a communication event (COM) occurs (COMG bit set or rising edge detected on TRGI, depending on the CCUS bit).

*Note: This bit acts only on channels that have a complementary output.*
### 13.4.3 TIM1 slave mode control register (TIM1_SMCR)

Address offset: 0x08  
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
</table>
| 15  | ETP: External trigger polarity | This bit selects whether ETR or 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. |
| 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. |
| 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 |
| 7:6  | ETF[3:0]: External trigger frequency |  
| 5:4  | MSM: Mode selection mode |  
| 3:2  | TS[2:0]: Trigger source |  
| 1    | OCCS: Output compare control select |  
| 0    | SMS[2:0]: Slave mode select |  

**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 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 consecutive 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: Care must be taken that \( f_{DTS} \) is replaced in the formula by \( CK\_INT \) when ETF[3:0] = 1, 2 or 3.*

**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: Reserved
- 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 44: TIMx Internal trigger connection 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  **OCCS:** OCREF clear selection.
This bit is used to select the OCREF clear source.
- 0: OCREF_CLR_INT is connected to the OCREF_CLR input
- 1: OCREF_CLR_INT is connected to ETRF

Bits 2:0  **SMS[2:0]:** 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 TI1FP1 edge depending on TI2FP2 level.
- 010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 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.

**Note:** The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

<table>
<thead>
<tr>
<th>Slave TIM</th>
<th>ITR0 (TS = 000)</th>
<th>ITR2 (TS = 010)</th>
<th>ITR3 (TS = 011)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM1</td>
<td>TIM15</td>
<td>TIM3</td>
<td>TIM17</td>
</tr>
</tbody>
</table>
13.4.4 TIM1 DMA/interrupt enable register (TIM1_DIER)

Address offset: 0x0C
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>TDE: Trigger DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Trigger DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Trigger DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>COMDE: COM DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: COM DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: COM DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>CC4DE: Capture/Compare 4 DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: CC4 DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: CC4 DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>CC3DE: Capture/Compare 3 DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: CC3 DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: CC3 DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CC2DE: Capture/Compare 2 DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: CC2 DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: CC2 DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CC1DE: Capture/Compare 1 DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: CC1 DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: CC1 DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>UDE: Update DMA request enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Update DMA request disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Update DMA request enabled</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BIE: Break interrupt enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Break interrupt disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Break interrupt enabled</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TIE: Trigger interrupt enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Trigger interrupt disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Trigger interrupt enabled</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>COMIE: COM interrupt enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: COM interrupt disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: COM interrupt enabled</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CC4IE: Capture/Compare 4 interrupt enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: CC4 interrupt disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: CC4 interrupt enabled</td>
<td></td>
</tr>
</tbody>
</table>
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 TIM1 status register (TIM1_SR)

Address offset: 0x10
Reset value: 0x0000

| Bit 15:13 | Reserved, must be kept at reset value. |
| 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, must be kept at reset value. |
| 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). 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 by writing it to '0'. |
|          | 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 matches the content of the TIMx_CCR1 register.
When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF
bit goes high on the counter overflow (in upcounting and up/down-counting modes) or
underflow (in downcounting mode)

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 repetition
counter = 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 13.4.3: TIM1 slave mode
     control register (TIM1_SMCR)), if URS=0 and UDIS=0 in the TIMx_CR1 register.

### 13.4.6  TIM1 event generation register (TIM1_EGR)

Address offset: 0x14
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bits 15:8  Reserved, must be kept at reset value.

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).
13.4.7 TIM1 capture/compare mode register 1 (TIM1_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 one must take care that the same bit can have a different meaning for the input stage and for the output stage.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>OC2CE: Output Compare 2 clear enable</td>
</tr>
<tr>
<td>14:12</td>
<td>OC2M[2:0]: Output Compare 2 mode</td>
</tr>
<tr>
<td>11</td>
<td>OC2PE: Output Compare 2 preload enable</td>
</tr>
<tr>
<td>10</td>
<td>OC2FE: Output Compare 2 fast enable</td>
</tr>
<tr>
<td>9:8</td>
<td>CC2S[1:0]: Capture/Compare 2 selection</td>
</tr>
<tr>
<td></td>
<td>This bit-field defines the direction of the channel (input/output) as well as the used input.</td>
</tr>
<tr>
<td></td>
<td>00: CC2 channel is configured as output</td>
</tr>
<tr>
<td></td>
<td>01: CC2 channel is configured as input, IC2 is mapped on TI2</td>
</tr>
<tr>
<td></td>
<td>10: CC2 channel is configured as input, IC2 is mapped on TI1</td>
</tr>
<tr>
<td></td>
<td>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)</td>
</tr>
<tr>
<td></td>
<td>Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCRER).</td>
</tr>
<tr>
<td>7</td>
<td>OC1CE: Output Compare 1 clear enable</td>
</tr>
<tr>
<td></td>
<td>OC1CE: Output Compare 1 Clear Enable</td>
</tr>
<tr>
<td></td>
<td>0: OC1Ref is not affected by the ETRF Input</td>
</tr>
<tr>
<td></td>
<td>1: OC1Ref is cleared as soon as a High level is detected on ETRF input</td>
</tr>
</tbody>
</table>
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 (this mode is used to generate a timing base).
- **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.

3: On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the OC1M active bits take the new value from the preloaded bits only when a COM event is generated.
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: 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).*

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).*
13.4.8 TIM1 capture/compare mode register 2 (TIM1_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.
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).*
Advanced-control timers (TIM1)

13.4.9 TIM1 capture/compare enable register (TIM1_CCER)

Address offset: 0x20

Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC4P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC3NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC4E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC3NE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC3P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC2NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC3E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC2NE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC2P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC2E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1NE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1E</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:14 Reserved, must be kept at reset value.

- **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

**CC1 channel configuration as output:**
- 0: OC1N active high.
- 1: OC1N active low.

**CC1 channel configuration as input:**
This bit is used in conjunction with CC1P to define the polarity of TI1FP1 and TI2FP1. Refer to CC1P description.

*Note:* On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1NP active bit takes the new value from the preloaded bits only when a Commutation event is generated.

*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.

*Note:* On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1NE active bit takes the new value from the preloaded bits only when a Commutation event is generated.

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:**
CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

- 00: non-inverted/rising edge
  The circuit is sensitive to TI1FP1 rising edge (capture or trigger operations in reset, external clock or trigger mode), TI1FP1 is not inverted (trigger operation in gated mode or encoder mode).
- 01: inverted/falling edge
  The circuit is sensitive to TI1FP1 falling edge (capture or trigger operations in reset, external clock or trigger mode), TI1FP1 is inverted (trigger operation in gated mode or encoder mode).
- 10: reserved, do not use this configuration.
- 11: non-inverted/both edges
  The circuit is sensitive to both TI1FP1 rising and falling edges (capture or trigger operations in reset, external clock or trigger mode), TI1FP1 is not inverted (trigger operation in gated mode). This configuration must not be used in encoder mode.

*Note:* On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1P active bit takes the new value from the preloaded bits only when a Commutation event is generated.

*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.

*Note:* On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1E active bit takes the new value from the preloaded bits only when a Commutation event is generated.

### Table 45. Output control bits for complementary OCx and OCxN channels with break feature

<table>
<thead>
<tr>
<th>Control bits</th>
<th>Output states(1)</th>
<th>OCx output state</th>
<th>OCxN output state</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOE bit</td>
<td>OSSI bit</td>
<td>OSSR bit</td>
<td>CCxE bit</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 45. Output control bits for complementary OCx and OCxN channels with break feature (continued)

<table>
<thead>
<tr>
<th>Control bits</th>
<th>Output states(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOE bit</td>
<td>OSSI bit</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 registers.

### 13.4.10 TIM1 counter (TIM1_CNT)

**Address offset:** 0x24

**Reset value:** 0x0000

<table>
<thead>
<tr>
<th>CNT[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **CNT[15:0]: Counter value**
13.4.11  TIM1 prescaler (TIM1_PSC)
Address offset: 0x28
Reset value: 0x0000

<table>
<thead>
<tr>
<th>PSC[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
</tr>
</tbody>
</table>

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”).

13.4.12  TIM1 auto-reload register (TIM1_ARR)
Address offset: 0x2C
Reset value: 0xFFFF

<table>
<thead>
<tr>
<th>ARR[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0  **ARR[15:0]:** Auto-reload 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 227 for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

13.4.13  TIM1 repetition counter register (TIM1_RCR)
Address offset: 0x30
Reset value: 0x0000

<table>
<thead>
<tr>
<th>REP[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
</tr>
</tbody>
</table>

---

292/775  RM0360 Rev 5
Bits 15:8  Reserved, must be kept at reset value.

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 reload 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.

### 13.4.14 TIM1 capture/compare register 1 (TIM1_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).

### 13.4.15 TIM1 capture/compare register 2 (TIM1_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 |

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).

### 13.4.16 TIM1 capture/compare register 3 (TIM1_CCR3)

Address offset: 0x3C

Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

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).
13.4.17 TIM1 capture/compare register 4 (TIM1_CCR4)

Address offset: 0x40
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15:0 CCR4[15:0]: Capture/Compare value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>If channel CC4 is configured as output:</strong></td>
</tr>
<tr>
<td>CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC4 output.</td>
</tr>
<tr>
<td><strong>If channel CC4 is configured as input:</strong></td>
</tr>
<tr>
<td>CCR4 is the counter value transferred by the last input capture 4 event (IC4).</td>
</tr>
</tbody>
</table>

13.4.18 TIM1 break and dead-time register (TIM1_BDTR)

Address offset: 0x44
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15: MOE: Main output enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: OC and OCN outputs are disabled or forced to idle state.</td>
</tr>
<tr>
<td>1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx_CCER register).</td>
</tr>
<tr>
<td>See OC/OCN enable description for more details (Section 13.4.9: TIM1 capture/compare enable register (TIM1_CCER) on page 288).</td>
</tr>
<tr>
<td><strong>Bit 14: AOE: Automatic output enable</strong></td>
</tr>
<tr>
<td>0: MOE can be set only by software</td>
</tr>
<tr>
<td>1: MOE can be set by software or automatically at the next update event (if the break input is not active)</td>
</tr>
<tr>
<td>Note: This bit cannot be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).</td>
</tr>
</tbody>
</table>
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).

*Note:* Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 12 **BKE**: Break enable
- 0: Break inputs (BRK and CCS clock failure event) disabled
- 1: Break inputs (BRK and CCS clock failure event) enabled

*Note:* This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

*Note:* Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

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 13.4.9: TIM1 capture/compare enable register (TIM1_CCER) on page 288).

- 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.

*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 13.4.9: TIM1 capture/compare enable register (TIM1_CCER) on page 288).

- 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.

*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 setup
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_dig with t_dig=t_DTS.
- DTG[7:5]=10x => DT=(64+DTG[5:0])x_t_dig with t_dig=2x_t_DTS.
- DTG[7:5]=110 => DT=(32+DTG[4:0])x_t_dig with t_dig=8x_t_DTS.
- DTG[7:5]=111 => DT=(32+DTG[4:0])x_t_dig with t_dig=16x_t_DTS.

Example if T_DTS=125 ns (8 MHz), 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 63 us 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).

### 13.4.19 **TIM1 DMA control register (TIM1_DCR)**

Address offset: 0x48
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>12</td>
<td>DBL[4:0]</td>
<td>rw</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>4</td>
<td>DBA[4:0]</td>
<td>rw</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>rw</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:8 **DBL[4:0]:** DMA burst length
This 5-bit vector defines the number of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address)
- 00000: 1 transfer
- 00001: 2 transfers
- 00010: 3 transfers
- ...
- 10001: 18 transfers

Bits 7:5 Reserved, must be kept at reset value.

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,
- ...

*Example:* Let us consider the following transfer: DBL = 7 transfers and DBA = TIMx_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx_CR1 address.
13.4.20 TIM1 DMA address for full transfer (TIM1_DMAR)

Address offset: 0x4C
Reset value: 0x0000

Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers (x = 2, 3, 4) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
   - DMA channel peripheral address is the DMAR register address
   - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
   - Number of data to transfer = 3 (See note below).
   - Circular mode disabled.

2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:
   DBL = 3 transfers, DBA = 0xE.

3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).

4. Enable TIMx

5. Enable the DMA channel

Note: This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let’s take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.
# 13.4.21 TIM1 register map

TIM1 registers are mapped as 16-bit addressable registers as described in the table below:

| Offset | Register   | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|--------|------------|--------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0x00   | TIM1_CR1   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value|        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x04   | TIM1_CR2   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value|        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x08   | TIM1_SMCR  |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value|        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x0C   | TIM1_DIER  |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value|        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x10   | TIM1_SR    |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value|        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x14   | TIM1_EGR   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value|        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x18   | TIM1_CCMR1 |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Output compare mode | OC2CE | OC2M | OC2P | OC2F | CC2S | CC2I | OC1M | OC1P | OC1F | OC1FE | OC1FE | CC1S | CC1I |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x1C   | TIM1_CCMR2 |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Output compare mode | OC4CE | OC4M | OC4P | OC4F | CC4S | CC4I | OC3M | OC3P | OC3F | OC3FE | OC3FE | CC3S | CC3I |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x20   | TIM1_CCR1  |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Input capture mode | IC2F | IC2M | IC2P | IC2F | CC2S | CC2I | IC1M | IC1P | IC1F | IC1FE | IC1FE | CC1S | CC1I |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x24   | TIM1_CNT   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | CNT[15:0]             |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x28   | TIM1_PSC   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | PSC[15:0]                |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x2C   | TIM1_ARR   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | ARR[15:0]                 |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |
| 0x30   | TIM1_RCR   |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | REP[7:0]                  |        |        |       |       |       |       |       |       |       |       |       |       |       |
|        | Reset value |        |        |       |       |       |       |       |       |       |       |       |       |       |

Table 46. TIM1 register map and reset values
Refer to Section 2.2 on page 37 for the register boundary addresses.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Offset</th>
<th>Register</th>
<th>Offset</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x34</td>
<td>TIM1_CCR1</td>
<td>0x38</td>
<td>TIM1_CCR2</td>
<td>0x3C</td>
<td>TIM1_CCR3</td>
</tr>
<tr>
<td>Reset value</td>
<td>CCR1[15:0]</td>
<td>Reset value</td>
<td>CCR2[15:0]</td>
<td>Reset value</td>
<td>CCR3[15:0]</td>
</tr>
<tr>
<td>0x40</td>
<td>TIM1_CCR4</td>
<td>0x44</td>
<td>TIM1_BDTR</td>
<td>0x48</td>
<td>TIM1_DCR</td>
</tr>
<tr>
<td>Reset value</td>
<td>CCR4[15:0]</td>
<td>Reset value</td>
<td></td>
<td>Reset value</td>
<td>DBL[4:0]</td>
</tr>
<tr>
<td>0x4C</td>
<td>TIM1_DMAR</td>
<td></td>
<td></td>
<td>Reset value</td>
<td></td>
</tr>
<tr>
<td>Reset value</td>
<td>DMAB[15:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14 General-purpose timers (TIM3)

14.1 TIM3 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 14.3.15.

14.2 TIM3 main features

General-purpose TIMx timer features include:

• 16-bit (TIM3) up, down, up/down auto-reload counter.
• 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
• Up to 4 independent channels for:
  – Input capture
  – Output compare
  – PWM generation (Edge- and Center-aligned modes)
  – One-pulse mode output
• Synchronization circuit to control the timer with external signals and to interconnect several timers.
• 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 circuitry for positioning purposes
• Trigger input for external clock or cycle-by-cycle current management
14.3 TIM3 functional description

14.3.1 Time-base unit

The main block of the programmable timer is a 16-bit/32-bit counter with its related auto-reload register. The counter can count up but also 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/32-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 94* and *Figure 95* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 94. Counter timing diagram with prescaler division change from 1 to 2**

![Counter Timing Diagram]

The diagram shows:
- CK_PSC: The clock input to the prescaler.
- CEN: The counter enable signal.
- Timer clock = CK_CNT: The clock signal that drives the counter.
- Counter register: The current value of the counter.
- Update event (UEV): The event that triggers the update.
- Prescaler control register: The prescaler ratio control register.
- Prescaler buffer: The buffer for the prescaler register.
- Prescaler counter: The actual counter value after the prescaler division change.
14.3.2 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 96. Counter timing diagram, internal clock divided by 1**

- **CK_PSC**
- **CNT_EN**
- Timer clock = CK_CNT
- Counter register
- Counter overflow
- Update event (UEV)
- Update interrupt flag (UIF)

**Figure 97. Counter timing diagram, internal clock divided by 2**

- **CK_PSC**
- **CNT_EN**
- Timer clock = CK_CNT
- Counter register
- Counter overflow
- Update event (UEV)
- Update interrupt flag (UIF)
Figure 98. Counter timing diagram, internal clock divided by 4

- **CK_PSC**
- **CNT_EN**
- Timer clock = CK_CNT
- Counter register: 0035 0036 0000 0001
- Counter overflow
- Update event (UEV)
- Update interrupt flag (UIF)

Figure 99. Counter timing diagram, internal clock divided by N

- **CK_PSC**
- Timer clock = CK_CNT
- Counter register: 1F 20 00
- Counter overflow
- Update event (UEV)
- Update interrupt flag (UIF)
Figure 100. Counter timing diagram, Update event when ARPE=0  
(TIMx_ARR not preloaded)

Figure 101. 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 generated 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 102. Counter timing diagram, internal clock divided by 1*
Figure 103. Counter timing diagram, internal clock divided by 2

Figure 104. Counter timing diagram, internal clock divided by 4
Figure 105. Counter timing diagram, internal clock divided by N

Figure 106. 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.

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are not equal to '00'. The Output compare interrupt flag of channels configured in output is set when: the counter counts down (Center aligned mode 1, CMS = "01"), the counter counts up (Center aligned mode 2, CMS = "10") the counter counts up and down (Center aligned mode 3, CMS = "11").

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 107. 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 14.4.1: TIM3 control register 1 (TIM3_CR1) on page 343).

Figure 108. Counter timing diagram, internal clock divided by 2
Figure 109. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36

1. Center-aligned mode 2 or 3 is used with an UIF on overflow.

Figure 110. Counter timing diagram, internal clock divided by N
Figure 111. Counter timing diagram, Update event with ARPE=1 (counter underflow)

CK_PSC

Timer clock = CK_CNT

Counter register

Counter underflow

Update event (UEV)

Update interrupt flag (UIF)

Auto-reload preload register

Auto-reload active register

Figure 112. Counter timing diagram, Update event with ARPE=1 (counter overflow)

CK_PSC

Timer clock = CK_CNT

Counter register

Counter overflow

Update event (UEV)

Update interrupt flag (UIF)

Auto-reload preload register

Auto-reload active register
14.3.3 Clock sources

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, Timer 1 can be configured to act as a prescaler for Timer 2. Refer to : Using one timer as prescaler for another on page 337 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 113 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

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.
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 and CC2NP=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.

For code example refer to the Appendix section A.8.2: Up counter on each 2 ETR rising edges.

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.
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 116 gives an overview of the external trigger input block.

For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

1. Counter clock = CK_CNT = CK_PSC
2. Counter register
3. Write TIF=0
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 117. Control circuit in external clock mode 2**

14.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an 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).
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 118. Capture/compare channel (example: channel 1 input stage)**

**Figure 119. Capture/compare channel 1 main circuit**
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.

14.3.5 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 it is written with 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 appropriate input filter duration in relation with the signal connected 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 most 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 the CC1P and CC1NP bits to 0 in the TIMx_CCMER 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_CCMER 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.

For code example refer to the Appendix section A.8.3: Input capture configuration.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- CC1F 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.

For code example refer to the Appendix section A.8.4: Input capture data management.

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.
14.3.6 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, one 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 to ‘0’ and the CC1NP 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’ and the CC2NP bit to ‘0’ (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.

For code example refer to the Appendix section A.8.5: PWM input configuration.

![Figure 121. PWM input mode timing](image-url)
14.3.7 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, one just needs 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.

14.3.8 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 CCxE and/or CCxDE bits if an interrupt and/or a DMA request is to be generated.
4. Select the output mode. For example, one 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.

For code example refer to the Appendix section A.8.7: Output compare configuration.
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 122.

**Figure 122. Output compare mode, toggle on OC1**

<table>
<thead>
<tr>
<th>TIMx_CNT</th>
<th>0039</th>
<th>003A</th>
<th>003B</th>
<th>B200</th>
<th>B201</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMx_CCR1</td>
<td>003A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OC1REF = OC1

Write 0xB201 in the CC1R register

Match detected on CCR1
Interrupt generated if enabled

**14.3.9 PWM mode**

Pulse width modulation mode allows 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. The corresponding preload register must be enabled 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, all registers must be initialized 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 forces the PWM by software while the timer is 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 304.*

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 123* shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

For code example refer to the Appendix section *A.8.9: Center-aligned PWM configuration example.*

![Figure 123. Edge-aligned PWM waveforms (ARR=8)](image-url)
Downcounting configuration

Downcounting is active when DIR bit in TIMx_CR1 register is high. Refer to Downcounting mode on page 308.

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 311.

Figure 124 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.
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 a value greater than the auto-reload value is written in the counter (TIMx_CNT> TIMx_ARR). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if 0 or the TIMx_ARR value is written 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.
14.3.10 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. One-pulse mode is selected 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.

For example one may want to generate a positive pulse on OC1 with a length of \( t_{\text{PULSE}} \) and after a delay of \( t_{\text{DELAY}} \) as soon as a positive edge is detected on the TI2 input pin.

Use TI2FP2 as trigger 1:

- Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP='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\textsubscript{DELAY} is defined by the value written in the TIMx_CCR1 register.
- The t\textsubscript{PULSE} is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1 + 1).
- Let’s say one 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 PWM mode 2 must be enabled by writing OC1M=111 in the TIMx_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE=1 in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case one has 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.

For code example refer to the Appendix section \textit{A.8.16: One-Pulse mode}.

Since only 1 pulse (Single mode) is needed, a 1 must be written 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). When OPM bit in the TIMx_CR1 register is set to ‘0’, so the Repetitive Mode is selected.

\textbf{Particular case: OCx fast enable}

In One-pulse mode, the edge detection on Ti\textsubscript{x} 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\textsubscript{DELAY} min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set 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.

For code example refer to the part of code, conditioned by \textit{PULSE\_WITHOUT\_DELAY} > 0 in the Appendix section \textit{A.8.16: One-Pulse mode}.

14.3.11 \textbf{Clearing the OCxREF signal on an external event}

1. The external trigger prescaler should be kept off: bits ETPS[1:0] in the TIMx_SMCR register are cleared to 00.
2. The external clock mode 2 must be disabled: bit ECE in the TIM1_SMCR register is cleared to 0.
3. The external trigger polarity (ETP) and the external trigger filter (ETF) can be configured according to the application’s needs.

For code example refer to the Appendix section \textit{A.8.10: ETR configuration to clear OCxREF}.

\textit{Figure 126} shows the behavior of the OCxREF signal when the ETRF input becomes high, for both values of the OCxCE enable bit. In this example, the timer TIMx is programmed in PWM mode.
1. In case of a PWM with a 100% duty cycle (if CCRx>ARR), OCxREF is enabled again at the next counter overflow.

14.3.12 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. CC1NP and CC2NP must be kept cleared. When needed, the input filter can be programmed as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to Table 47. 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 the TIMx_ARR must be configured 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 do not switch at the same time.

Table 47. Counting direction versus encoder signals

<table>
<thead>
<tr>
<th>Active edge</th>
<th>Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)</th>
<th>TI1FP1 signal</th>
<th>TI2FP2 signal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rising</td>
<td>Falling</td>
<td>Rising</td>
</tr>
<tr>
<td>Counting on TI1 only</td>
<td>High</td>
<td>Down</td>
<td>Up</td>
</tr>
<tr>
<td>Counting on TI2 only</td>
<td>High</td>
<td>No Count</td>
<td>No Count</td>
</tr>
<tr>
<td>Counting on TI1 and TI2</td>
<td>High</td>
<td>Down</td>
<td>Up</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Up</td>
<td>Down</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>No Count</td>
<td>No Count</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Up</td>
<td>Down</td>
</tr>
</tbody>
</table>

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.

Figure 127 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, TI2FP2 mapped on TI2)
- CC1P = 0, CC1NP = ’0’ (TIMx_CCER register, TI1FP1 noninverted, TI1FP1 = TI1)
- CC2P = 0, CC2NP = ’0’ (TIMx_CCER register, TI2FP2 noninverted, TI2FP2 = TI2)
- SMS = 011 (TIMx_SMCR register, both inputs are active on both rising and falling edges)
- CEN = 1 (TIMx_CR1 register, Counter is enabled)

For code example refer to the Appendix section A.8.10: ETR configuration to clear OCxREF.
Figure 127. Example of counter operation in encoder interface mode

Figure 128 gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P=1).

Figure 128. 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. Dynamic information can be obtained (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. This can be done 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.

14.3.13 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 13.3.18 on page 266.

14.3.14 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 do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx_CCMR1 register. Write CC1P=0 and CC1NP=0 in TIMx_CCRER 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.

For code example refer to the Appendix section A.8.12: Reset mode.

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 129. Control circuit in reset mode](image-url)
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 do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx_CCMR1 register. Write CC1P=1 and CC1NP=0 in TIMx_CCRER 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).

For code example refer to the Appendix section A.8.13: Gated mode.

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.

![Control circuit in gated mode](image)

**Figure 130. Control circuit in gated mode**

1. The configuration "CCxP=CCxNP=1" (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.

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 do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. CC2S bits are selecting the input capture source only, CC2S=01 in TIMx_CCMR1 register. Write
CC2P=1 and CC2NP=0 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.

For code example refer to the Appendix section 8.14: Trigger mode.

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 131. Control circuit in trigger mode**

<table>
<thead>
<tr>
<th>T12</th>
<th>cnt_en</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Counter clock = ck_cnt = ck_psc

Counter register

TIF

 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=01 in TIMx_CCMR1 register to select only the input capture source
   - CC1P=0 and CC1NP=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.

   For code example refer to the Appendix section A.8.15: External clock mode 2 + trigger mode.

   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 132. Control circuit in external clock mode 2 + trigger mode**

14.3.15 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.
Using one timer as prescaler for another

For example, Timer 1 can be configured to act as a prescaler for Timer 3. Refer to Figure 133. To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each update event UEV. If MMS=010 is written 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 3, Timer 3 must be configured in slave mode using ITR1 as internal trigger. This is selected through the TS bits in the TIM3_SMCR register (writing TS=000).
- Then the Timer2's slave mode controller should be configured in external clock mode 1 (write SMS=111 in the TIM3_SMCR register). This causes Timer 3 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 within their respective TIMx_CR1 registers. Make sure to enable Timer2 before enabling Timer1.

For code example refer to the Appendix section A.8.17: Timer prescaling another timer.

Note: If OCx is selected on Timer 1 as trigger output (MMS=1xx), its rising edge is used to clock the counter of timer 3.

Using one timer to enable another timer

In this example, we control the enable of Timer 3 with the output compare 1 of Timer 1. Refer to Figure 133 for connections. Timer 3 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 ($f_{\text{CK\_CNT}} = f_{\text{CK\_INT}}/3$).

- 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_CCMRx register).
- Configure Timer 3 to get the input trigger from Timer 1 (TS=000 in the TIM3_SMCR register).
- Configure Timer 3 in gated mode (SMS=101 in TIM3_SMCR register).
- Enable Timer 3 by writing ‘1’ in the CEN bit (TIM3_CR1 register).
- Start Timer 1 by writing ‘1’ in the CEN bit (TIM1_CR1 register).

For code example refer to the Appendix section A.8.18: Timer enabling another timer.

**Note:** The counter 3 clock is not synchronized with counter 1, this mode only affects the Timer 3 counter enable signal.

**Figure 134. Gating timer 3 with OC1REF of timer 1**

In the example in Figure 134, the Timer 3 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. Then any value can be written 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 3. Timer 1 is the master and starts from 0. Timer 3 is the slave and starts from 0x7E. The prescaler ratio is the same for both
timers. Timer 3 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 Counter Enable signal (CNT_EN) as a trigger output (MMS=001 in the TIM1_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1_CCMR1 register).
- Configure Timer 3 to get the input trigger from Timer 1 (TS=000 in the TIM3_SMCR register).
- Configure Timer 3 in gated mode (SMS=101 in TIM3_SMCR register).
- Reset Timer 1 by writing ‘1’ in UG bit (TIM1_EGR register).
- Reset Timer 3 by writing ‘1’ in UG bit (TIM3_EGR register).
- Initialize Timer 3 to 0xE7 by writing ‘0xE7’ in the timer 3 counter (TIM3_CNTL).
- Enable Timer 3 by writing ‘1’ in the CEN bit (TIM3_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).

For code example refer to the Appendix section A.8.19: Master and slave synchronization.

**Figure 135. Gating timer 3 with Enable of timer 1**
Using one timer to start another timer

In this example, we set the enable of Timer 3 with the update event of Timer 1. Refer to Figure 133 for connections. Timer 3 starts counting from its current value (which can be nonzero) on the divided internal clock as soon as the update event is generated by Timer 1. When Timer 3 receives the trigger signal its CEN bit is automatically set and the counter counts until we write 0 to the CEN bit in the TIM3_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 3 to get the input trigger from Timer 1 (TS=000 in the TIM3_SMCR register).
- Configure Timer 3 in trigger mode (SMS=110 in TIM3_SMCR register).
- Start Timer 1 by writing 1 in the CEN bit (TIM1_CR1 register).

Figure 136. Triggering timer 3 with update of timer 1

As in the previous example, both counters can be initialized before starting counting. Figure 137 shows the behavior with the same configuration as in Figure 136 but in trigger mode instead of gated mode (SMS=110 in the TIM3_SMCR 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 3 with the enable of Timer 1. Refer to Figure 133 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 3):

- 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 3 to get the input trigger from Timer 1 (TS=000 in the TIM3_SMCR register).
- Configure Timer 3 in trigger mode (SMS=110 in the TIM3_SMCR register).

For code example refer to the Appendix section A.8.20: Two timers synchronized by an external trigger.

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 an offset can easily be inserted between them by writing any of the counter registers (TIMx_CNT). One can see that the master/slave mode insert a delay between CNT_EN and CK_PSC on timer 1.
14.3.16 Debug mode

When the microcontroller enters debug mode (Arm® Cortex®-M0 core - halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBGMCU module.
14.4 **TIM3 registers**

Refer to *Section 1.2 on page 33* for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

### 14.4.1 TIM3 control register 1 (TIM3_CR1)

Address offset: 0x00

Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CKD[1:0]</td>
<td>ARPE</td>
<td>CMS[1:0]</td>
<td>DIR</td>
<td>OPM</td>
<td>URS</td>
<td>UDIS</td>
<td>CEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:10  Reserved, must be kept at reset value.

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[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 alternately. 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 alternately. 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 alternately. 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.
### 14.4.2 TIM3 control register 2 (TIM3_CR2)

Address offset: 0x04  
Reset value: 0x0000

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

Bits 15:8 Reserved, must be kept at reset value.

**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 13.3.18: Interfacing with Hall sensors on page 266

**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, must be kept at reset value.
14.4.3 TIM3 slave mode control register (TIM3_SMCR)

Address offset: 0x08
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETP:</td>
<td>External trigger polarity</td>
</tr>
<tr>
<td>ECE:</td>
<td>External clock enable</td>
</tr>
<tr>
<td>ETPS[1:0]</td>
<td>External trigger prescaler</td>
</tr>
<tr>
<td>ETF[3:0]</td>
<td></td>
</tr>
<tr>
<td>MSM</td>
<td></td>
</tr>
<tr>
<td>TS[2:0]</td>
<td></td>
</tr>
<tr>
<td>OCCS</td>
<td></td>
</tr>
<tr>
<td>SMS[2:0]</td>
<td></td>
</tr>
</tbody>
</table>

Bit 15 **ETP:** External trigger polarity
- This bit selects whether ETR or \( \overline{ETR} \) is used for trigger operations
- 0: ETR is noninverted, 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.
- 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 \( CK_{INT} \) 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 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 consecutive 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: Care must be taken that \( f_{DTS} \) is replaced in the formula by \( f_{CK\_INT} \) when ETF[3:0] = 1, 2 or 3.

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).
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 48: TIM3 internal trigger connection 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 OCCS: OCREF clear selection.
This bit is used to select the OCREF clear source.
0: OCREF_CLR_INT is connected to the OCREF_CLR input
1: OCREF_CLR_INT is connected to ETRF
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 TI1FP1 edge depending on TI2FP2 level.
010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 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.

Note: The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

<table>
<thead>
<tr>
<th>Slave TIM</th>
<th>ITR0 (TS = 000)</th>
<th>ITR2 (TS = 010)</th>
<th>ITR3 (TS = 011)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM3</td>
<td>TIM1</td>
<td>TIM15</td>
<td>TIM14</td>
</tr>
</tbody>
</table>

14.4.4 TIM3 DMA/Interrupt enable register (TIM3_DIER)

Address offset: 0x0C
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDE</td>
<td>CC4DE</td>
<td>CC3DE</td>
<td>CC2DE</td>
<td>CC1DE</td>
<td>UDE</td>
<td>TIE</td>
<td>CC4IE</td>
<td>CC3IE</td>
<td>CC2IE</td>
<td>CC1IE</td>
<td>UIE</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bit 15 Reserved, must be kept at reset value.

Bit 14 TDE: Trigger DMA request enable
0: Trigger DMA request disabled.
1: Trigger DMA request enabled.

Bit 13 Reserved, must be kept at reset value.

Bit 12 CC4DE: Capture/Compare 4 DMA request enable
0: CC4 DMA request disabled.
1: CC4 DMA request enabled.
14.4.5 TIM3 status register (TIM3_SR)

Address offset: 0x10

Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 11</th>
<th>CC3DE: Capture/Compare 3 DMA request enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC3 DMA request disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC3 DMA request enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 10</th>
<th>CC2DE: Capture/Compare 2 DMA request enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC2 DMA request disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC2 DMA request enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 9</th>
<th>CC1DE: Capture/Compare 1 DMA request enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC1 DMA request disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC1 DMA request enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 8</th>
<th>UDE: Update DMA request enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Update DMA request disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Update DMA request enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Bit 6</th>
<th>TIE: Trigger interrupt enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trigger interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Trigger interrupt enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 5</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Bit 4</th>
<th>CC4IE: Capture/Compare 4 interrupt enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC4 interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC4 interrupt enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>CC3IE: Capture/Compare 3 interrupt enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC3 interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC3 interrupt enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>CC2IE: Capture/Compare 2 interrupt enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC2 interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC2 interrupt enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>CC1IE: Capture/Compare 1 interrupt enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CC1 interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>CC1 interrupt enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>UIE: Update interrupt enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Update interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Update interrupt enabled.</td>
</tr>
</tbody>
</table>

---

**14.4.5 TIM3 status register (TIM3_SR)**

Address offset: 0x10

Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC4OF</td>
<td>CC3OF</td>
<td>CC2OF</td>
<td>CC1OF</td>
<td>TIF</td>
<td>CC4IF</td>
<td>CC3IF</td>
<td>CC2IF</td>
<td>CC1IF</td>
<td>UIF</td>
<td>CC4OF</td>
<td>CC3OF</td>
<td>CC2OF</td>
<td>CC1OF</td>
<td>TIF</td>
<td>CC4IF</td>
</tr>
<tr>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
</tr>
</tbody>
</table>

---
Bits 15:13  Reserved, must be kept at reset value.
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, must be kept at reset value.
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. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.
0: No trigger event occurred
1: Trigger interrupt pending
Bit 5  Reserved, must be kept at reset value.
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 matches the content of the TIMx_CCR1 register.  
When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit  
goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow  
(in downcounting mode)  
**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 and if 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.
### 14.4.6 TIM3 event generation register (TIM3_EGR)

Address offset: 0x14  
Reset value: 0x0000

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:7 Reserved, must be kept at reset value.

- **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, must be kept at reset value.

- **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**:  
      The 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).
14.4.7 TIM3 capture/compare mode register 1 (TIM3_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, OCx describes its function when the channel is configured in output, ICx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

### 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_CCRER).

- **Bit 7 OC1CE:** Output compare 1 clear 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
### Input capture mode

**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.(this mode is used to generate a timing base).  
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:** 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).

**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).
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 consecutive 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: Care must be taken that 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).*
14.4.8  TIM3 capture/compare mode register 2 (TIM3_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

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).*

14.4.9 **TIM3 capture/compare enable register (TIM3_CCER)**

Address offset: 0x20

Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>CC4NP</th>
<th>Bit 14</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 13</td>
<td>CC4P</td>
<td>Bit 12</td>
<td>CC4E</td>
</tr>
<tr>
<td>Bit 11</td>
<td>CC3NP</td>
<td>Bit 10</td>
<td>Reserved</td>
</tr>
<tr>
<td>Bit 9</td>
<td>CC3P</td>
<td>Bit 8</td>
<td>CC3E</td>
</tr>
<tr>
<td>Bit 7</td>
<td>CC2NP</td>
<td>Bit 6</td>
<td>CC2E</td>
</tr>
<tr>
<td>Bit 5</td>
<td>CC2P</td>
<td>Bit 4</td>
<td>CC1NP</td>
</tr>
<tr>
<td>Bit 3</td>
<td>CC1P</td>
<td>Bit 2</td>
<td>CC1E</td>
</tr>
</tbody>
</table>

**Bit 15** **CC4NP**: Capture/Compare 4 output Polarity.
Refer to CC1NP description

**Bit 14** Reserved, must be kept at reset value.

**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 output Polarity.
Refer to CC1NP description

**Bit 10** Reserved, must be kept at reset value.

**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 output Polarity.
Refer to CC1NP description

Bit 6  Reserved, must be kept at reset value.

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 output Polarity.

**CC1 channel configured as output:**
CC1NP must be kept cleared in this case.

**CC1 channel configured as input:**
This bit is used in conjunction with CC1P to define Ti1FP1/Ti2FP1 polarity. refer to CC1P description.

Bit 2  Reserved, must be kept at reset value.

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:**
CC1NP/CC1P bits select Ti1FP1 and Ti2FP1 polarity for trigger or capture operations.
00: noninverted/rising edge
Circuit is sensitive to TixFP1 rising edge (capture, trigger in reset, external clock or trigger mode), TixFP1 is not inverted (trigger in gated mode, encoder mode).
01: inverted/falling edge
Circuit is sensitive to TixFP1 falling edge (capture, trigger in reset, external clock or trigger mode), TixFP1 is inverted (trigger in gated mode, encoder mode).
10: reserved, do not use this configuration.
11: noninverted/both edges
Circuit is sensitive to both TixFP1 rising and falling edges (capture, trigger in reset, external clock or trigger mode), TixFP1 is not inverted (trigger in gated mode). This configuration must not be used for encoder mode.

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>
<thead>
<tr>
<th><strong>Table 49. Output control bit for standard OCx channels</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CCxE bit</strong></td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Note: The state of the external IO pins connected to the standard OCx channels depends on the OCx channel state and the GPIO registers.

14.4.10 TIM3 counter (TIM3_CNT)
Address offset: 0x24
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 CNT[15:0]: Low counter value

14.4.11 TIM3 prescaler (TIM3_PSC)
Address offset: 0x28
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 PSC[15:0]: Prescaler value
The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event.

14.4.12 TIM3 auto-reload register (TIM3_ARR)
Address offset: 0x2C
Reset value: 0xFFFF FFFF

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 ARR[15:0]: Low Auto-reload value
ARR is the value to be loaded in the actual auto-reload register.
Refer to Section 14.3.1: Time-base unit on page 302 for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.
14.4.13 TIM3 capture/compare register 1 (TIM3_CCR1)
Address offset: 0x34
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **CCR1[15:0]**: Low 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). Otherwise 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).

14.4.14 TIM3 capture/compare register 2 (TIM3_CCR2)
Address offset: 0x38
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **CCR2[15:0]**: Low 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 signaled on OC2 output.
- **If channel CC2 is configured as input:**
  - CCR2 is the counter value transferred by the last input capture 2 event (IC2).

14.4.15 TIM3 capture/compare register 3 (TIM3_CCR3)
Address offset: 0x3C
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
Bits 15:0 \textbf{CCR3}[15:0]: Low Capture/Compare 3 value

\textbf{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 signaled on OC3 output.

\textbf{If channel CC3 is configured as input:}

CCR3 is the counter value transferred by the last input capture 3 event (IC3).
### 14.4.16 TIM3 capture/compare register 4 (TIM3_CCR4)

Address offset: 0x40  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0</td>
<td><strong>CCR4[15:0]</strong>: Low Capture/Compare 4 value</td>
</tr>
<tr>
<td></td>
<td>1. If CC4 channel is configured as output (CC4S bits):</td>
</tr>
<tr>
<td></td>
<td>CCR4 is the value to be loaded in the actual capture/compare 4 register</td>
</tr>
<tr>
<td></td>
<td>(preload value). It is loaded permanently if the preload feature is</td>
</tr>
<tr>
<td></td>
<td>not selected in the TIMx_CCMR4 register (bit OC4PE). Otherwise, the preload</td>
</tr>
<tr>
<td></td>
<td>value is copied in the active capture/compare 4 register when an update</td>
</tr>
<tr>
<td></td>
<td>event occurs. The active capture/compare register contains the value to</td>
</tr>
<tr>
<td></td>
<td>be compared to the counter TIMx_CNT and signaled on OC4 output.</td>
</tr>
<tr>
<td></td>
<td>2. If CC4 channel is configured as input (CC4S bits in TIMx_CCMR4 register):</td>
</tr>
<tr>
<td></td>
<td>CCR4 is the counter value transferred by the last input capture 4 event</td>
</tr>
<tr>
<td></td>
<td>(IC4).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-13</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>
| 12-8| **DBL[4:0]**: DMA burst length  
|     | This 5-bit vector defines the number of DMA transfers (the timer recognizes |
|     | a burst transfer when a read or a write access is done to the TIMx_DMAR  |
|     | address).  
|     | 00000: 1 transfer,  
|     | 00001: 2 transfers,  
|     | 00010: 3 transfers,  
|     | ...  
|     | 10001: 18 transfers. |
| 7-5 | Reserved, must be kept at reset value. |
| 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,  
|     | 00010: TIMx_CR2,  
|     | 00011: TIMx_SMCR,  
|     | ...  
|     | **Example**: Let us consider the following transfer: DBL = 7 transfers & DBA = TIMx_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx_CR1 address. |
14.4.18 TIM3 DMA address for full transfer (TIM3_DMAR)

Address offset: 0x4C
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>DMAB[15:0]</th>
<th>DMA register for burst accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw</td>
<td>A read or write operation to the DMAR register accesses the register located at the address (TIMx_CR1 address) + (DBA + DMA index) x 4 where 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 automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx_DCR).</td>
</tr>
</tbody>
</table>
Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers (x = 2, 3, 4) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
   - DMA channel peripheral address is the DMAR register address
   - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
   - Number of data to transfer = 3 (See note below).
   - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows: DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

For code example refer to the Appendix section **A.8.20: Two timers synchronized by an external trigger**.

**Note:** This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let us take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.
### 14.4.19 TIM3 register map

TIM3 registers are mapped as described in the table below:

**Table 50. TIM3 register map and reset values**

| Offset | Register    | Bits | 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   | TIM3_CR1    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x04   | TIM3_CR2    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x08   | TIM3_SMCR   |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x0C   | TIM3_DIER   |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x10   | TIM3_SR     |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x14   | TIM3_EGR    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x18   | TIM3_CCMR1  |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Output compare mode | |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x1C   | TIM3_CCMR2  |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Output compare mode | |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x20   | TIM3_CCER   |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x24   | TIM3_CNT    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | CNT[15:0]   |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x28   | TIM3_PSC    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | PSC[15:0]   |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x2C   | TIM3_ARR    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | ARR[15:0]   |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x30   | Reserved    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
Table 50. TIM3 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   |
|--------|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0x34   | TIM3_CCR1 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x38   | TIM3_CCR1 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x3C   | TIM3_CCR1 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x40   | TIM3_CCR1 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x44   | Reserved   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x48   | TIM3_DCR   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x4C   | TIM3_DMAR  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

Refer to Section 2.2 on page 37 for the register boundary addresses.
15 Basic timer (TIM6/TIM7)

This section applies to STM32F030x8, STM32F070xB and STM32F030xC devices only. TIM7 is available only on STM32F070xB and STM32F030xC devices.

15.1 TIM6/TIM7 introduction

The basic timer TIM6 consists of a 16-bit auto-reload counter driven by a programmable prescaler.

15.2 TIM6/TIM7 main features

- 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
- Interrupt/DMA generation on the update event: counter overflow

Figure 139. Basic timer block diagram
15.3 **TIM6/TIM7 functional description**

15.3.1 **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 140* and *Figure 141* give some examples of the counter behavior when the prescaler ratio is changed on the fly.
Figure 140. Counter timing diagram with prescaler division change from 1 to 2

Figure 141. Counter timing diagram with prescaler division change from 1 to 4
15.3.2 Counter modes

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 142. Counter timing diagram, internal clock divided by 1**
Figure 143. Counter timing diagram, internal clock divided by 2

Figure 144. Counter timing diagram, internal clock divided by 4
Figure 145. Counter timing diagram, internal clock divided by N

Figure 146. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded)
Figure 147. Counter timing diagram, update event when ARPE=1
(TIMx_ARR preloaded)
15.3.3 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 148* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

*Figure 148. Control circuit in normal mode, internal clock divided by 1*

![Diagram](image)

15.3.4 Debug mode

When the microcontroller enters the debug mode (Cortex™-M0 core - halted), the TIMx counter either continues to work normally or stops, depending on the DBG_TIMx_STOP configuration bit in the DBG module.
15.4 TIM6/TIM7 registers

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

15.4.1 TIM6/TIM7 control register 1 (TIMx_CR1)

Address offset: 0x00
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>ARPE</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>OPM</td>
<td>URS</td>
<td>UDIS</td>
<td>CEN</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

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.
15.4.2 TIM6/TIM7 DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:9  Reserved, must be kept at reset value.

Bit 8  **UDE**: Update DMA request enable
0: Update DMA request disabled.
1: Update DMA request enabled.

Bits 7:1  Reserved, must be kept at reset value.

Bit 0  **UIE**: Update interrupt enable
0: Update interrupt disabled.
1: Update interrupt enabled.
15.4.3 TIM6/TIM7 status register (TIMx_SR)

Address offset: 0x10
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:1 Reserved, must be kept at reset value.

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.

15.4.4 TIM6/TIM7 event generation register (TIMx_EGR)

Address offset: 0x14
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:1 Reserved, must be kept at reset value.

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).

15.4.5 TIM6/TIM7 counter (TIMx_CNT)

Address offset: 0x24
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **CNT[15:0]**: Counter value
15.4.6 TIM6/TIM7 prescaler (TIMx_PSC)

Address offset: 0x28
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **PSC[15:0]**: Prescaler value
The counter clock frequency $f_{CK\_CNT}$ 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.

15.4.7 TIM6/TIM7 auto-reload register (TIMx_ARR)

Address offset: 0x2C
Reset value: 0xFFFF

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **ARR[15:0]**: Auto-reload value
ARR is the value to be loaded into the actual auto-reload register.
Refer to Section 15.3.1: Time-base unit on page 368 for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.
15.4.8 TIM6/TIM7 register map

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

Table 51. TIM6/TIM7 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   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | ARPE|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 0   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x0C   | TIMx_DIER  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 0   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x10   | TIMx_SR    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 0   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x14   | TIMx_EGR   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 0   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x24   | TIMx_CNT   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | CNT[15:0]|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 0    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x28   | TIMx_PSC   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | PSC[15:0]|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 0    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x2C   | TIMx_ARR   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | ARR[15:0]|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Reset value|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     | 1    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

Refer to Section 2.2 on page 37 for the register boundary addresses.
16 General-purpose timer (TIM14)

16.1 TIM14 introduction

The TIM14 general-purpose timer consists 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).

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 TIM14 timer is completely independent, and does not share any resources. It can be synchronized together as described in Section 14.3.15.

16.2 TIM14 main features

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65535 (can be changed “on the fly”)
- independent channel for:
  - Input capture
  - Output compare
  - PWM generation (edge-aligned mode)
- Interrupt generation on the following events:
  - Update: counter overflow, counter initialization (by software)
  - Input capture
  - Output compare
16.3 TIM14 functional description

16.3.1 Time-base unit

The main block of the programmable general-purpose 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. 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 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 151* and *Figure 152* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 150. Counter timing diagram with prescaler division change from 1 to 2**

**Figure 151. Counter timing diagram with prescaler division change from 1 to 4**

16.3.2 Counter operation

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.
Setting the UG bit in the TIMx_EGR register 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 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 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 152. Counter timing diagram, internal clock divided by 1**

[Diagram showing the timing for internal clock divided by 1]

**Figure 153. Counter timing diagram, internal clock divided by 2**

[Diagram showing the timing for internal clock divided by 2]
Figure 154. Counter timing diagram, internal clock divided by 4

Figure 155. Counter timing diagram, internal clock divided by N

Figure 156. Counter timing diagram, update event when ARPE=0
(TIMx_ARR not preloaded)
16.3.3 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 158 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

16.3.4 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).
**Figure 159** to **Figure 161** 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 159. 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 160. Capture/compare channel 1 main circuit**
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.

### 16.3.5 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 it is written with 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:

1. 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 mode and the TIMx_CCR1 register becomes read-only.

2. Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TIx (ICxF bits in the TIMx_CCMRx register). Let us imagine that, when toggling, the input signal is not stable during at most 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.

3. Select the edge of the active transition on the TI1 channel by programming CC1P and CC1NP bits to ‘00’ in the TIMx_CCR1 register (rising edge in this case).

4. 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).

5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCR1 register.

6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_DIER register.

For code example refer to the Appendix section A.8.3: Input capture configuration.

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 CC1E bit.

For code example refer to the Appendix section A.8.4: Input capture data management.

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 requests can be generated by software by setting the corresponding CCxG bit in the TIMx_EGR register.

### 16.3.6 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, one just needs 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.

The comparison between the TIMx_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt requests can be sent accordingly. This is described in the output compare mode section below.

### 16.3.7 Output compare mode

This function is used to control an output waveform or to indicate when a period of time has elapsed.

When a match is found between the capture/compare register and the counter, the output compare function:
1. 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_CCRx 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.

2. Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).

3. Generates an interrupt if the corresponding interrupt mask is set (CCxIE bit in the TIMx_DIER register).

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.

For code example refer to the Appendix section A.8.7: Output compare configuration.

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 162.
16.3.8 PWM mode

Pulse Width Modulation mode allows 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. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register 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, all registers must be initialized by setting the UG bit in the TIMx_EGR register.

The OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. The 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_CNT ≤ TIMx_CCRx.

The timer is able to generate PWM in edge-aligned mode only since the counter is upcounting.

**PWM edge-aligned mode**

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 163 shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.
16.3.9 Debug mode

When the microcontroller enters debug mode (Cortex™-M0 core halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module.

16.4 TIM14 registers

16.4.1 TIM14 control register 1 (TIM14_CR1)

Address offset: 0x00
Reset value: 0x0000
Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 CKD[1:0]: 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, Tlx).

00: tDTS = tCK_INT
01: tDTS = 2 × tCK_INT
10: tDTS = 4 × tCK_INT
11: Reserved

Bit 7 ARPE: Auto-reload preload enable
0: TIMx_ARR register is not buffered
1: TIMx_ARR register is buffered

Bits 6:3 Reserved, must be kept at reset value.

Bit 2 URS: Update request source
This bit is set and cleared by software to select the update interrupt (UEV) sources.
0: Any of the following events generate an UEV if enabled:
  – Counter overflow
  – Setting the UG bit
1: Only counter overflow generates an UEV if enabled.

Bit 1 UDIS: Update disable
This bit is set and cleared by software to enable/disable update interrupt (UEV) event generation.
0: UEV enabled. An UEV is generated by one of the following events:
  – Counter overflow
  – Setting the UG bit.
  Buffered registers are then loaded with their preload values.
1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

Bit 0 CEN: Counter enable
0: Counter disabled
1: Counter enabled

### 16.4.2 TIM14 interrupt enable enable register (TIM14_DIER)

Address offset: 0xOC
Reset value: 0x0000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | CC1IE | UIE |

Bits 15:2 Reserved, must be kept at reset value.

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
### 16.4.3 TIM14 status register (TIM14_SR)

Address offset: 0x10
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:10</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>9</td>
<td>CC1OF: Capture/Compare 1 overcapture flag</td>
</tr>
<tr>
<td></td>
<td>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'.</td>
</tr>
<tr>
<td></td>
<td>0: No overcapture has been detected.</td>
</tr>
<tr>
<td></td>
<td>1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set</td>
</tr>
<tr>
<td>8:2</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>1</td>
<td>CC1IF: Capture/compare 1 interrupt flag</td>
</tr>
<tr>
<td></td>
<td>Condition: channel CC1 is configured as output</td>
</tr>
<tr>
<td></td>
<td>This flag is set by hardware when the counter matches the compare value.</td>
</tr>
<tr>
<td></td>
<td>It is cleared by software.</td>
</tr>
<tr>
<td></td>
<td>0: No match.</td>
</tr>
<tr>
<td></td>
<td>1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow.</td>
</tr>
<tr>
<td></td>
<td>Condition: channel CC1 is configured as input</td>
</tr>
<tr>
<td></td>
<td>This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.</td>
</tr>
<tr>
<td></td>
<td>0: No input capture occurred.</td>
</tr>
<tr>
<td></td>
<td>1: The counter value has been captured in TIMx_CCR1 register (an edge has been detected on IC1 which matches the selected polarity).</td>
</tr>
<tr>
<td>0</td>
<td>UIF: Update interrupt flag</td>
</tr>
<tr>
<td></td>
<td>This bit is set by hardware on an update event. It is cleared by software.</td>
</tr>
<tr>
<td></td>
<td>0: No update occurred.</td>
</tr>
<tr>
<td></td>
<td>1: Update interrupt pending. This bit is set by hardware when the registers are updated:</td>
</tr>
<tr>
<td></td>
<td>– At overflow and if UDIS='0' in the TIMx_CR1 register.</td>
</tr>
<tr>
<td></td>
<td>– 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.</td>
</tr>
</tbody>
</table>

### 16.4.4 TIM14 event generation register (TIM14_EGR)

Address offset: 0x14
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>CC1G: Capture/compare 1 group flag</td>
</tr>
<tr>
<td>8</td>
<td>UG: Update group flag</td>
</tr>
<tr>
<td>7:2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Update group pending flag</td>
</tr>
</tbody>
</table>

Note: Bits 15:10 are reserved and must be kept at reset value, bit 9 is the Capture/Compare 1 overcapture flag, bit 8 is the Capture/Compare 1 interrupt flag, and bit 0 is the Update interrupt flag.
16.4.5 TIM14 capture/compare mode register 1 [alternate] (TIM14_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). 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 one must take care that the same bit can have a different meaning for the input stage and for the output stage.

<table>
<thead>
<tr>
<th>Bit 15:8</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 14:2</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>Bit 1</td>
<td>CC1G: Capture/compare 1 generation</td>
</tr>
<tr>
<td></td>
<td>This bit is set by software in order to generate an event, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>1: A capture/compare event is generated on channel 1:</td>
</tr>
<tr>
<td></td>
<td><strong>Condition: channel CC1 is configured as output</strong></td>
</tr>
<tr>
<td></td>
<td>CC1IF flag is set, Corresponding interrupt or is sent if enabled.</td>
</tr>
<tr>
<td></td>
<td><strong>Condition: channel CC1 is configured as input</strong></td>
</tr>
<tr>
<td></td>
<td>The current value of the counter is captured in TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.</td>
</tr>
<tr>
<td>Bit 0</td>
<td>UG: Update generation</td>
</tr>
<tr>
<td></td>
<td>This bit can be set by software, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
16.4.6 TIM14 capture/compare mode register 1 [alternate] (TIM14_CCMR1)

Address offset: 0x18
Reset value: 0x0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). 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 one must take care that the same bit can have a different meaning for the input stage and for the output stage.
Bits 15:7  Reserved, must be kept at reset value.

Bits 6:4  **OC1M[2:0]**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on CC1P bit.

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 - Channel 1 is active as long as TIMx_CNT < TIMx_CCR1 else inactive.

111: PWM mode 2 - Channel 1 is inactive as long as TIMx_CNT < TIMx_CCR1 else active.

*Note:* In PWM mode 1 or 2, the OCREF level changes when the result of the comparison changes or when the output compare mode switches from frozen 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:* 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. OC is then set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0  **CC1S[1:0]**: 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: Reserved

11: Reserved

*Note:* CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

### 16.4.7 TIM14 capture/compare enable register (TIM14_CCER)

Address offset: 0x20

Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1P</td>
<td>CC1E</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

396/775  RM0360 Rev 5
Bits 15:4  Reserved, must be kept at reset value.

Bit 3  **CC1NP**: Capture/Compare 1 complementary output Polarity.
CC1 channel configured as output: CC1NP must be kept cleared.
CC1 channel configured as input: CC1NP bit is used in conjunction with CC1P to define T11FP1 polarity (refer to CC1P description).

Bit 2  Reserved, must be kept at reset value.

Bit 1  **CC1P**: Capture/Compare 1 output Polarity.

**Condition: CC1 channel configured as output**
- 0: OC1 active high
- 1: OC1 active low

**Condition: CC1 channel configured as input**
The CC1P bit selects T11FP1 and T12FP1 polarity for trigger or capture operations.
- 00: noninverted/rising edge
- 01: inverted/falling edge
- 10: reserved, do not use this configuration.
- 11: noninverted/both edges

Bit 0  **CC1E**: Capture/Compare 1 output enable.

**Condition: CC1 channel configured as output**:
- 0: Off - OC1 is not active
- 1: On - OC1 signal is output on the corresponding output pin

**Condition: 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>
<thead>
<tr>
<th>CCxE bit</th>
<th>OCx output state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Output Disabled (OCx=’0’, OCx_EN=’0’)</td>
</tr>
<tr>
<td>1</td>
<td>OCx=OCxREF + Polarity, OCx_EN=’1’</td>
</tr>
</tbody>
</table>

**Table 52. Output control bit for standard OCx channels**

*Note: The state of the external I/O pins connected to the standard OCx channels depends on the OCx channel state and the GPIO registers.*

**16.4.8 TIM14 counter (TIM14_CNT)**

Address offset: 0x24
Reset value: 0x0000

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CNT[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0 **CNT[15:0]**: Counter value
### 16.4.9 TIM14 prescaler (TIM14_PSC)

Address offset: 0x28  
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

**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.

### 16.4.10 TIM14 auto-reload register (TIM14_ARR)

Address offset: 0x2C  
Reset value: 0xFFFF

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

**Bits 15:0 ARR[15:0]: Auto-reload value**

ARR is the value to be loaded in the actual auto-reload register.

Refer to the Section 16.3.1: Time-base unit on page 381 for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 16.4.11 TIM14 capture/compare register 1 (TIM14_CCR1)

Address offset: 0x34  
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

**Bits 15:0 CCR1[15:0]: Capture/Compare 1 value**

**Condition: 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.

**Condition: channel CC1 is configured as input**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).
16.4.12 TIM14 option register (TIM14_OR)
Address offset: 0x50
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:2 Reserved, must be kept at reset value.

Bits 1:0 **TI1_RMP[1:0]**: Timer Input 1 remap
Set and cleared by software.
00: TIM14 Channel1 is connected to the GPIO. Refer to the alternate function mapping in the device datasheets.
01: TIM14 Channel1 is connected to the RTCCLK.
10: TIM14 Channel1 is connected to the HSE/32 Clock.
11: TIM14 Channel1 is connected to the microcontroller clock output (MCO), this selection is controlled by the MCO[2:0] bits of the Clock configuration register (RCC_CFGR) (see Section 7.4.2: Clock configuration register (RCC_CFGR)).

16.4.13 TIM14 register map
TIM14 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 53. TIM14 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   | TIM14_CR1         | Res |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |                   | Res |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x08   | Reserved          |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x0C   | TIM14_DIER        |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |                   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x10   | TIM14_SR          |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |                   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x14   | TIM14_EGR         |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |                   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x18   | TIM14_CCMR1       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        | Output compare mode |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |                   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x1C   | Reserved          |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|        |                   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

Reset value
### Table 53. TIM14 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  |
|--------|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0x20   | TIM14_CCER    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x24   | TIM14_CNT     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x28   | TIM14_PSC     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x2C   | TIM14_ARR     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x30   | Reserved      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x34   | TIM14_CCR1    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x38 to 0x4C | Reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x50   | TIM14_OR      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Refer to Section 2.2 on page 37 for the register boundary addresses.
17 General-purpose timers (TIM15/16/17)

TIM15 is not available on STM32F030x4 and STM32F030x6 devices.

17.1 TIM15/16/17 introduction

The TIM15/16/17 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, 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 TIM15/16/17 timers are completely independent, and do not share any resources. The TIM15 can be synchronized with other timers.

17.2 TIM15 main features

TIM15 includes the following features:

- 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
- Up to 2 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge-aligned mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time (for channel 1 only)
- 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 the reset state or a known state
- Interrupt/DMA generation on the following events:
  - Update: counter overflow, 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 (interrupt request)
Figure 164. TIM15 block diagram

- General-purpose timers (TIM15/16/17)
- RM0360
- 402/775 RM0360 Rev 5

Notes:
- Preload registers transferred to active registers on U event according to control bit
- Event
- Interrupt & DMA output
17.3 **TIM16 and TIM17 main features**

The TIM16 and TIM17 timers include the following features:

- 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
- One channel for:
  - Input capture
  - Output compare
  - PWM generation (Edge-aligned mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time
- 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 the reset state or a known state
- Interrupt/DMA generation on the following events:
  - Update: counter overflow
  - Input capture
  - Output compare
  - Break input
17.4 TIM15/16/17 functional description

17.4.1 Time-base unit

The main block of the programmable general purpose 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)
- 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 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 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 151* and *Figure 152* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 166. Counter timing diagram with prescaler division change from 1 to 2**

<table>
<thead>
<tr>
<th>CK_PSC</th>
<th>CEN</th>
<th>Timer = CK_CNT</th>
<th>Counter register</th>
<th>Update event (UEV)</th>
<th>Prescaler control register</th>
<th>Prescaler buffer</th>
<th>Prescaler counter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>F7 F8 F9 FA FB FC</td>
<td>00 01 02 03</td>
<td>0</td>
<td>0</td>
<td>0 1 0 1 0 1</td>
</tr>
</tbody>
</table>

Write a new value in TIMx_PSC
17.4.2 Counter operation

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 168. Counter timing diagram, internal clock divided by 1**

![Counter timing diagram](image-url)
Figure 169. Counter timing diagram, internal clock divided by 2

Figure 170. Counter timing diagram, internal clock divided by 4
Figure 171. Counter timing diagram, internal clock divided by N

Figure 172. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded)
17.4.3 Repetition counter

Section 16.3.1: 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.

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx_RCR register value (refer to Figure 174). 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.
17.4.4 Clock sources

The counter clock can be provided by the following clock sources:

- **Internal clock (CK_INT)**
- **External clock mode1: external input pin (only for TIM15)**
- **Internal trigger inputs (ITRx) (only for TIM15): using one timer as the prescaler for another timer, for example, TIM1 can be configured to act as a prescaler for TIM15. Refer to Using one timer as prescaler for another for more details.**

**Internal clock source (CK_INT)**

For TIM15 if the slave mode controller is disabled (SMS=000), then the CEN 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 16.3.4 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

**Figure 175. 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 176. 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_CCR 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 it does not need to be configured.

For code example refer to the Appendix section A.8.1: Upcounter on TI2 rising edge.

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 177. Control circuit in external clock mode 1**

17.4.5 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

*Figure 159* to *Figure 181* 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).
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 179. Capture/compare channel 1 main circuit**
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.
17.4.6 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 it is written with '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 appropriate input filter duration in relation with the signal connected 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 most 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 fDTS 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.

For code example refer to the Appendix section A.8.3: Input capture configuration.

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.

For code example refer to the Appendix section A.8.4: Input capture data management.

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.
17.4.7 PWM input mode (only for TIM15)

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, one can measure the period (in TIMx_CCR1 register) and the duty cycle (in TIMx_CCR2 register) of the PWM applied on T11 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’ and the CC2NP bit to ‘0’ (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.

For code example refer to the Appendix section A.8.5: PWM input configuration.

![Figure 182. PWM input mode timing](image_url)

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.

17.4.8 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, one just needs 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.

17.4.9 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 (OCXMX=000), be set active (OCXMX=001), be set inactive (OCXMX=010) or can toggle (OCXMX=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.

For code example refer to the Appendix section A.8.2: Up counter on each 2 ETR rising edges.

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 162.

**Figure 183. Output compare mode, toggle on OC1**

17.4.10 **PWM mode**

Pulse Width Modulation mode allows 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. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register 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, all registers must be initialized 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.

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 163 shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.
17.4.11 Complementary outputs and dead-time insertion

The TIM15/16/17 general-purpose timers can output one complementary signal and manage the switching-off and switching-on of the outputs.

This time is generally known as dead-time and it has to be adjusted depending on the devices that are connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

The polarity of the outputs (main output OCx or complementary OCxN) can be selected 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 55: Output control bits for complementary OCx and OCxN channels with break feature on page 443 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 185. Complementary output with dead-time insertion**

![Figure 185 Complementary output with dead-time insertion](MS31095V1)

**Figure 186. Dead-time waveforms with delay greater than the negative pulse**

![Figure 186 Dead-time waveforms with delay greater than the negative pulse](MS31096V1)

**Figure 187. Dead-time waveforms with delay greater than the positive pulse**

![Figure 187 Dead-time waveforms with delay greater than the positive pulse](MS31097V1)
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 17.5.16: TIM15 break and dead-time register (TIM15_BDTR) on page 446 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 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.

17.4.12 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 55: Output control bits for complementary OCx and OCxN channels with break feature on page 443 for more details.

The source for break (BRK) channel can be an external source connected to the BKIN pin or one of the following internal sources:

- the core LOCKUP output
- the PVD output
- the SRAM parity error signal
- a clock failure event generated by the CSS detector

When exiting from reset, the break circuit is disabled and the MOE bit is low. The break function can be enabled 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. When the BKE and BKP bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

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 MOE is set to 1 whereas it was low, a delay must be inserted (dummy instruction) before reading it correctly. This is because the write acts on the asynchronous signal whereas the read reflects 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.
- 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 it is written with 1 again. In this case, it can be used for security and the break input can be connected 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 to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). The protection can be selected among 3 levels with the LOCK bits in the TIMx_BDTR register. Refer to Section 17.5.16: TIM15 break and dead-time register (TIM15_BDTR) on page 446. The LOCK bits can be written only once after an MCU reset.

The Figure 188 shows an example of behavior of the outputs in response to a break.
Figure 188. Output behavior in response to a break
17.4.13 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. One-pulse mode is selected 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: \( CNT < CCRx \leq ARR \) (in particular, \( 0 < CCRx \))
- In downcounting: \( CNT > CCRx \)

For example one may want to generate a positive pulse on OC1 with a length of \( t_{\text{PULSE}} \) and after a delay of \( t_{\text{DELAY}} \) 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).

For code example refer to the Appendix section A.8.16: One-Pulse mode.
The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

- The $t_{\text{DELAY}}$ is defined by the value written in the TIMx_CCR1 register.
- The $t_{\text{PULSE}}$ is defined by the difference between the auto-reload value and the compare value ($\text{TIMx\_ARR} - \text{TIMx\_CCR1}$).
- Let's say one wants 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 PWM mode 2 must be enabled by writing OC1M=111 in the TIMx_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE='1' in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case one has 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.

Since only 1 pulse is needed, a 1 must be written 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 $t_{\text{DELAY}}$ we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set 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.

For code example refer to the part of code conditioned by PULSE WITHOUT DELAY > 0 in the Appendix section A.8.16: One-Pulse mode.

### 17.4.14 TIM15 external trigger synchronization

This section applies to STM32F030x8, STM32F070xB and STM32F030xC devices only.

The TIM15 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 do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. 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.

For code example refer to the Appendix section A.8.12: Reset mode.

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.
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 do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx_CCMR1 register. Write CC1P=1 in TIMx_CCRER 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).

For code example refer to the Appendix section \textit{A.8.13: Gated mode}.

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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure191.png}
\caption{Control circuit in gated mode}
\end{figure}

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 do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. 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.

For code example refer to the Appendix section A.8.14: Trigger mode.

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 192. Control circuit in trigger mode**

![Control circuit in trigger mode](image)

The TIM timers are linked together internally for timer synchronization or chaining. Refer to Section 14.3.15: Timer synchronization on page 336 for details.

### 17.4.15 Timer synchronization (TIM15)

This section applies to STM32F030x8 STM32F070xB and STM32F030xC devices only.

The TIM timers are linked together internally for timer synchronization or chaining. Refer to Section 14.3.15: Timer synchronization on page 336 for details.

### 17.4.16 Debug mode

When the microcontroller enters debug mode (Cortex™-M0 core halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module.
17.5 TIM15 registers

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

17.5.1 TIM15 control register 1 (TIM15_CR1)

Address offset: 0x00
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:10 Reserved, must be kept at reset value.

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 (tDTS) used by the dead-time generators and the digital filters (Tlx)

- 00: tDTS = tCK_INT
- 01: tDTS = 2*tCK_INT
- 10: tDTS = 4*tCK_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 6:4 Reserved, must be kept at reset value.

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 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 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 and 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.

### 17.5.2 TIM15 control register 2 (TIM15_CR2)
Address offset: 0x04
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIS2</td>
<td>OIS1N</td>
<td>OIS1</td>
<td>MMS[2:0]</td>
<td>CCDS</td>
<td>CCUS</td>
<td>CCPC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:11 Reserved, must be kept at reset value.

Bit 10 **OIS2**: Output idle state 2 (OC2 output)
0: OC2=0 when MOE=0
1: OC2=1 when MOE=0

Note: This bit cannot be modified as long as LOCK level 1, 2 or 3 has been programmed
   (LOCK bits in TIMx_BKR register).

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 Reserved, must be kept at reset value.
17.5.3 TIM15 slave mode control register (TIM15_SMCR)

Address offset: 0x08
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 6:4 **MMS[2:0]**: 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.
100: **Compare** - OC1REF signal is used as trigger output (TRGO).
101: **Compare** - OC2REF 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 COMG bit only.
1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI.

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

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.*
Bits 15:8 Reserved, must be kept at reset value.

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: Reserved
- 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 (FI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)

See [Table 54: TIMx Internal trigger connection](#) 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, must be kept at reset value.

Bits 2:0 **SMS[2:0]**: 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.
- 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 54. TIMx Internal trigger connection

<table>
<thead>
<tr>
<th>Slave TIM</th>
<th>ITR1 (TS = 001)</th>
<th>ITR2 (TS = 010)</th>
<th>ITR3 (TS = 011)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM15</td>
<td>TIM3</td>
<td>TIM16_OC</td>
<td>TIM17_OC</td>
</tr>
</tbody>
</table>


**17.5.4 TIM15 DMA/interrupt enable register (TIM15_DIER)**

Address offset: 0x0C  
Reset value: 0x0000

```
<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>
```

- **Bit 15** Reserved, must be kept at reset value.
- **Bit 14** **TDE**: Trigger DMA request enable  
  0: Trigger DMA request disabled  
  1: Trigger DMA request enabled

- **Bits 13:11** Reserved, must be kept at reset value.
- **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

- **Bits 4:3** Reserved, must be kept at reset value.
- **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
17.5.5 TIM15 status register (TIM15_SR)

Address offset: 0x10
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:11</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CC2OF: Capture/Compare 2 overcapture flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refer to CC1OF description.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CC1OF: Capture/Compare 1 overcapture flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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'.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No overcapture has been detected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BIF: Break interrupt flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No break event occurred</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: An active level has been detected on the break input</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TIF: Trigger interrupt flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No trigger event occurred</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Trigger interrupt pending</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>COMIF: COM interrupt flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This flag is set by hardware on a COM event (once the capture/compare control bits –CCxE, CCxNE, OCxM– have been updated). It is cleared by software by writing it to '0'.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No COM event occurred</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: COM interrupt pending</td>
<td></td>
</tr>
<tr>
<td>4:3</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:11 Reserved, must be kept at reset value.
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, must be kept at reset value.
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 a COM event (once the capture/compare control bits –CCxE, CCxNE, OCxM– have been updated). It is cleared by software by writing it to ‘0’.
0: No COM event occurred
1: COM interrupt pending

Bits 4:3 Reserved, must be kept at reset value.
Bit 2  **CC2IF**: Capture/Compare 2 interrupt flag
refer to CC1IF description

Bit 1  **CC1IF**: Capture/Compare 1 interrupt flag

**Condition: channel CC1 is configured as output**
This flag is set by hardware when the counter matches the compare value, it is cleared by software.

0: No match.
1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register.
   When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**Condition: 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 regarding the repetition counter value (update if repetition counter = 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 17.5.3: TIM15 slave mode control register (TIM15_SMCR)), if URS=0 and UDIS=0 in the TIMx_CR1 register.
17.5.6 TIM15 event generation register (TIM15_EGR)

Address offset: 0x14
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15:8</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7</td>
<td><strong>BG</strong>: Break generation</td>
</tr>
<tr>
<td></td>
<td>This bit is set by software in order to generate an event, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.</td>
</tr>
<tr>
<td>Bit 6</td>
<td><strong>TG</strong>: Trigger generation</td>
</tr>
<tr>
<td></td>
<td>This bit is set by software in order to generate an event, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>1: The TIF flag is set in TIMx_SR register. Related interrupt or DMA transfer can occur if enabled.</td>
</tr>
<tr>
<td>Bit 5</td>
<td><strong>COMG</strong>: Capture/Compare control update generation</td>
</tr>
<tr>
<td></td>
<td>This bit can be set by software, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>1: When the CCPC bit is set, it is possible to update the CCxE, CCxNE and OCxM bits. Note: This bit acts only on channels that have a complementary output.</td>
</tr>
<tr>
<td>Bits 4:3</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>Bit 2</td>
<td><strong>CC2G</strong>: Capture/Compare 2 generation</td>
</tr>
<tr>
<td></td>
<td>Refer to CC1G description</td>
</tr>
<tr>
<td>Bit 1</td>
<td><strong>CC1G</strong>: Capture/Compare 1 generation</td>
</tr>
<tr>
<td></td>
<td>This bit is set by software in order to generate an event, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>1: A capture/compare event is generated on channel 1:</td>
</tr>
<tr>
<td></td>
<td><strong>Condition: channel CC1 is configured as output</strong></td>
</tr>
<tr>
<td></td>
<td>CC1IF flag is set. Corresponding interrupt or DMA request is sent if enabled.</td>
</tr>
<tr>
<td></td>
<td><strong>Condition: channel CC1 is configured as input</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Bit 0</td>
<td><strong>UG</strong>: Update generation</td>
</tr>
<tr>
<td></td>
<td>This bit can be set by software, it is automatically cleared by hardware.</td>
</tr>
<tr>
<td></td>
<td>0: No action</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
</tbody>
</table>
17.5.7  TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)

Address offset: 0x18
Reset value: 0x0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). 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 one must take care that the same bit can have a different meaning for the input stage and for the output stage.

<table>
<thead>
<tr>
<th>Bit 15-12</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC2F[3:0]</td>
<td>Input capture 2 filter</td>
</tr>
<tr>
<td>IC2PSC[1:0]</td>
<td>Input capture 2 prescaler</td>
</tr>
<tr>
<td>CC2S[1:0]</td>
<td>Capture/Compare 2 selection</td>
</tr>
</tbody>
</table>

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).
17.5.8 TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). 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 one must take care that the same bit can have a different meaning for the input stage and for the output stage.

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 consecutive events are needed to validate a transition on the output:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>No filter, sampling is done at fDTS</td>
</tr>
<tr>
<td>0001</td>
<td>fSAMPLING = fCK_INT, N = 2</td>
</tr>
<tr>
<td>0010</td>
<td>fSAMPLING = fCK_INT, N = 4</td>
</tr>
<tr>
<td>0011</td>
<td>fSAMPLING = fCK_INT, N = 8</td>
</tr>
<tr>
<td>0100</td>
<td>fSAMPLING = fDTS / 2, N = 6</td>
</tr>
<tr>
<td>0101</td>
<td>fSAMPLING = fDTS / 2, N = 8</td>
</tr>
<tr>
<td>0110</td>
<td>fSAMPLING = fDTS / 4, N = 6</td>
</tr>
<tr>
<td>0111</td>
<td>fSAMPLING = fDTS / 4, N = 8</td>
</tr>
<tr>
<td>1000</td>
<td>fSAMPLING = fDTS / 8, N = 6</td>
</tr>
<tr>
<td>1001</td>
<td>fSAMPLING = fDTS / 8, N = 8</td>
</tr>
<tr>
<td>1010</td>
<td>fSAMPLING = fDTS / 16, N = 5</td>
</tr>
<tr>
<td>1011</td>
<td>fSAMPLING = fDTS / 16, N = 6</td>
</tr>
<tr>
<td>1100</td>
<td>fSAMPLING = fDTS / 16, N = 8</td>
</tr>
<tr>
<td>1101</td>
<td>fSAMPLING = fDTS / 32, N = 5</td>
</tr>
<tr>
<td>1110</td>
<td>fSAMPLING = fDTS / 32, N = 6</td>
</tr>
<tr>
<td>1111</td>
<td>fSAMPLING = fDTS / 32, N = 8</td>
</tr>
</tbody>
</table>

Note: Care must be taken that fDTS is replaced in the formula by CK_INT when ICxF[3:0] = 1, 2 or 3.

Bits 3:2 IC1PSC[1:0]: 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).

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>no prescaler, capture is done each time an edge is detected on the capture input</td>
</tr>
<tr>
<td>01</td>
<td>capture is done once every 2 events</td>
</tr>
<tr>
<td>10</td>
<td>capture is done once every 4 events</td>
</tr>
<tr>
<td>11</td>
<td>capture is done once every 8 events</td>
</tr>
</tbody>
</table>

Bits 1:0 CC1S[1:0]: Capture/Compare 1 Selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>CC1 channel is configured as output</td>
</tr>
<tr>
<td>01</td>
<td>CC1 channel is configured as input, IC1 is mapped on TI1</td>
</tr>
<tr>
<td>10</td>
<td>CC1 channel is configured as input, IC1 is mapped on TI2</td>
</tr>
<tr>
<td>11</td>
<td>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)</td>
</tr>
</tbody>
</table>

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).
<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>14:12</td>
<td><strong>OC2M[2:0]</strong>: Output Compare 2 mode</td>
</tr>
<tr>
<td>11</td>
<td><strong>OC2PE</strong>: Output Compare 2 preload enable</td>
</tr>
<tr>
<td>10</td>
<td><strong>OC2FE</strong>: Output Compare 2 fast enable</td>
</tr>
<tr>
<td>9:8</td>
<td><strong>CC2S[1:0]</strong>: Capture/Compare 2 selection</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>6:4</td>
<td><strong>OC1M[2:0]</strong>: Output Compare 1 mode</td>
</tr>
</tbody>
</table>

**Bit 15**: Reserved, must be kept at reset value.

**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**: Reserved, must be kept at reset value.

**Bits 6:4**: **OC1M[2:0]**: 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*: **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*: 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: 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).*

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 of 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[1:0]**: 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).*

## 17.5.9 **TIM15 capture/compare enable register (TIM15_CCER)**

Address offset: 0x20
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:8</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>7</td>
<td><strong>CC2NP</strong>: Capture/Compare 2 complementary output polarity</td>
</tr>
<tr>
<td></td>
<td>Refer to CC1NP description.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>5</td>
<td><strong>CC2P</strong>: Capture/Compare 2 output polarity</td>
</tr>
<tr>
<td></td>
<td>Refer to CC1P description.</td>
</tr>
<tr>
<td>4</td>
<td><strong>CC2E</strong>: Capture/Compare 2 output enable</td>
</tr>
<tr>
<td></td>
<td>Refer to CC1E description.</td>
</tr>
</tbody>
</table>
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

**Condition: CC1 channel configured as output**

0: OC1 active high  
1: OC1 active low

**Condition: CC1 channel configured as input**

The CC1NP/CC1P bits select the polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

00: Noninverted/rising edge: circuit is sensitive to TIxFP1's rising edge (capture, trigger in reset or trigger mode), TIxFP1 is not inverted (trigger in gated mode).

01: Inverted/falling edge: circuit is sensitive to TIxFP1's falling edge (capture, trigger in reset, or trigger mode), TIxFP1 is inverted (trigger in gated mode).

10: Reserved, do not use this configuration.

11: Noninverted/both edges: circuit is sensitive to both the rising and falling edges of TIxFP1 (capture, trigger in reset or trigger mode), TIxFP1 is not inverted (trigger in gated mode).

*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

**Condition: 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.

**Condition: 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 55. Output control bits for complementary OCx and OCxN channels with break feature

<table>
<thead>
<tr>
<th>Control bits</th>
<th>Output states(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOE bit</td>
<td>OSSI bit</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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.
17.5.10  TIM15 counter (TIM15_CNT)
Address offset: 0x24
Reset value: 0x0000

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
```

Bits 15:0  **CNT[15:0]**: Counter value

17.5.11  TIM15 prescaler (TIM15_PSC)
Address offset: 0x28
Reset value: 0x0000

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
```

Bits 15:0  **PSC[15:0]**: Prescaler value
The counter clock frequency (CK_CNT) is equal to \( f_{\text{CK_PSC}} / (\text{PSC[15:0]} + 1) \).
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”).

17.5.12  TIM15 auto-reload register (TIM15_ARR)
Address offset: 0x2C
Reset value: 0xFFFF

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
```

Bits 15:0  **ARR[15:0]**: Auto-reload value
ARR is the value to be loaded in the actual auto-reload register.
Refer to the Section 16.3.1: Time-base unit on page 381 for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.
17.5.13 TIM15 repetition counter register (TIM15_RCR)

Address offset: 0x30
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:8 Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 7:0 REP[7:0]: Repetition counter value</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>It means in PWM mode (REP+1) corresponds to the number of PWM periods in edge-aligned mode.</td>
</tr>
</tbody>
</table>

17.5.14 TIM15 capture/compare register 1 (TIM15_CCR1)

Address offset: 0x34
Reset value: 0x0000

| Bits 15:0 CCR1[15:0]: Capture/Compare 1 value |
| Condition: 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. |
| Condition: if channel CC1 is configured as input |
| CCR1 is the counter value transferred by the last input capture 1 event (IC1). |

17.5.15 TIM15 capture/compare register 2 (TIM15_CCR2)

Address offset: 0x38
Reset value: 0x0000
17.5.16 TIM15 break and dead-time register (TIM15_BDTR)

Address offset: 0x44
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>MOE</td>
<td>Main output enable</td>
</tr>
<tr>
<td>14</td>
<td>AOE</td>
<td>Automatic output enable</td>
</tr>
<tr>
<td>13</td>
<td>BKP</td>
<td>Break polarity</td>
</tr>
<tr>
<td>12</td>
<td>BKE</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>OSSR</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>OSSI</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>LOCK[1:0]</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>DTG[7:0]</td>
<td></td>
</tr>
</tbody>
</table>

Note: As the bits AOE, BKP, BKE, OSSR, OSSI and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.

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_CCR register)
See OC/OCN enable description for more details (Section 17.5.9: TIM15 capture/compare enable register (TIM15_CCR) on page 441).

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).
Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.
Bit 12 **BKE**: Break enable

- 0: Break inputs (BRK and CCS clock failure event) disabled
- 1: Break inputs (BRK and CCS clock failure event) enabled

*Note:* This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

*Note:* Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

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 17.5.9: TIM15 capture/compare enable register (TIM15_CCER) on page 441).

- 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 cannot 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 17.5.9: TIM15 capture/compare enable register (TIM15_CCER) on page 441).

- 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 cannot 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.
### 17.5.17 TIM15 DMA control register (TIM15_DCR)

Address offset: 0x48  
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:13  Reserved, must be kept at reset value.

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).  
00000: 1 transfer  
00001: 2 transfers  
00010: 3 transfers  
...  
10001: 18 transfers  

Bits 7:5  Reserved, must be kept at reset value.

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  
...
17.5.18 TIM15 DMA address for full transfer (TIM15_DMAR)

Address offset: 0x4C
Reset value: 0x0000

Bits 15:0 DMAB[15:0]: DMA register for burst accesses
A read or write operation to the DMAR register accesses the register located at the address
(TMx_CR1 address) + (DBA + DMA index) x 4
where 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 automatically controlled by the
DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx_DCR).

17.5.19 TIM15 register map

TIM15 registers are mapped as 16-bit addressable registers as described in the table
below:

Table 56. TIM15 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   | TIM15_CR1  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |    |    |    |    |
| 0x04   | TIM15_CR2  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |    |    |    |    |
| 0x08   | TIM15_SMCR |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |    |    |    |    |
| 0x0C   | TIM15_DIER |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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   | TIM15_SR   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  | 0  |    |    |    |    |
| 0x14   | TIM15_EGR  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  | 0  | 0  | 0  |    |    |
| 0x18   | TIM15_CCMR1|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  | 0  | 0  | 0  |    |
| 0x20   | TIM15_CCER |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  | 0  | 0  | 0  | 0  |    |    |    |    |    |
### Table 56. TIM15 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  |
|--------|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0x24   | TIM15_CNT  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |
| 0x28   | TIM15_PSC  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |
| 0x2C   | TIM15_ARR  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value| 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
| 0x30   | TIM15_RCR  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x34   | TIM15_CCR1 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |
| 0x38   | TIM15_CCR2 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |
| 0x44   | TIM15_BDTR |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |
| 0x48   | TIM15_DCR  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  |
| 0x4C   | TIM15_DMAR |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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 Section 2.2 on page 37 for the register boundary addresses.

### 17.6 TIM16/TIM17 registers

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

#### 17.6.1 TIMx control register 1 (TIMx_CCR1)(x = 16 to 17)

Address offset: 0x00

Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKD[1:0]</td>
<td>ARPE</td>
<td>OPM</td>
<td>URS</td>
<td>UDIS</td>
<td>CEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

450/775 RM0360 Rev 5
Bits 15:10  Reserved, must be kept at reset value.

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 (tDTS) used by the dead-time generators and the digital filters (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, 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 6:4  Reserved, must be kept at reset value.

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
17.6.2 TIMx control register 2 (TIMx_CR2)(x = 16 to 17)

Address offset: 0x04
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:10 Reserved, must be kept at reset value.

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).*

Bits 7:4 Reserved, must be kept at reset value.

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 COMG bit only.
- 1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI.

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

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.*

17.6.3 TIMx DMA/interrupt enable register (TIMx_DIER)(x = 16 to 17)

Address offset: 0x0C
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
Bits 15:10 Reserved, must be kept at reset value.

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 Reserved, must be kept at reset value.

Bit 5 **COMIE**: COM interrupt enable
- 0: COM interrupt disabled
- 1: COM interrupt enabled

Bits 4:2 Reserved, must be kept at reset value.

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

### 17.6.4 TIMx status register (TIMx_SR)(x = 16 to 17)

Address offset: 0x10

Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1OF</td>
<td>BIF</td>
<td>COMIF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC1IF</td>
<td>UIF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rc_w0</td>
<td>rc_w0</td>
</tr>
</tbody>
</table>

Bits 15:10 Reserved, must be kept at reset value.

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, must be kept at reset value.

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  Reserved, must be kept at reset value.

Bit 5  **COMIF**: COM interrupt flag

This flag is set by hardware on a COM event (once the capture/compare control bits –CCxE, CCxNE, OCxM– have been updated). It is cleared by software by writing it to ’0’.

0: No COM event occurred
1: COM interrupt pending

Bits 4:2  Reserved, must be kept at reset value.

Bit 1  **CC1IF**: Capture/Compare 1 interrupt flag

**Condition: channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, it is cleared by software.

0: No match.
1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**Condition: 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 regarding the repetition counter value (update if repetition counter = 0)
   
   – 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.

### 17.6.5  TIMx event generation register (TIMx_EGR)(x = 16 to 17)

**Address offset**: 0x14

**Reset value**: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>COMG</td>
<td>CC1G</td>
<td>UG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Bits 15:8 Reserved, must be kept at reset value.

**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 Reserved, must be kept at reset value.**

**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 the CCPC bit is set, it is possible to update the CCxE, CCxNE and OCxM bits

*Note:* This bit acts only on channels that have a complementary output.

### Bits 4:2 Reserved, must be kept at reset value.

**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:

  **Condition: channel CC1 is configured as output:**
  
  CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

  **Condition: 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).

### 17.6.6 TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 16 to 17)

**Address offset:** 0x18

**Reset value:** 0x0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). 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 one must take care that the same bit can have a different meaning for the input stage and for the output stage.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Reserved, must be kept at reset value</td>
</tr>
<tr>
<td>14</td>
<td>reserved</td>
</tr>
<tr>
<td>13</td>
<td>reserved</td>
</tr>
<tr>
<td>12</td>
<td>reserved</td>
</tr>
<tr>
<td>11</td>
<td>reserved</td>
</tr>
<tr>
<td>10</td>
<td>reserved</td>
</tr>
<tr>
<td>9</td>
<td>reserved</td>
</tr>
<tr>
<td>8</td>
<td>reserved</td>
</tr>
<tr>
<td>7</td>
<td>IC1F[3:0]</td>
</tr>
<tr>
<td>6</td>
<td>IC1PSC[1:0]</td>
</tr>
<tr>
<td>5</td>
<td>CC1S[1:0]</td>
</tr>
<tr>
<td>4</td>
<td>rw</td>
</tr>
<tr>
<td>3</td>
<td>rw</td>
</tr>
<tr>
<td>2</td>
<td>rw</td>
</tr>
<tr>
<td>1</td>
<td>rw</td>
</tr>
<tr>
<td>0</td>
<td>rw</td>
</tr>
</tbody>
</table>

**STI**

RM0360 Rev 5 455/775
Bits 15:8 Reserved, must be kept at reset value.

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

Note: Care must be taken that f_DTS is replaced in the formula by CK_INT when ICxF[3:0] = 1, 2 or 3.

Bits 3:2 IC1PSC[1:0]: 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[1:0]: 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).

17.6.7 TIMx capture/compare mode register 1 [alternate]
(TIMx_CCMR1)(x = 16 to 17)
Address offset: 0x18
Reset value: 0x0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). 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 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
one must take care that the same bit can have a different meaning for the input stage and for the output stage.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-7</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>6-4</td>
<td><strong>OC1M[2:0]</strong>: Output Compare 1 mode</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>000: Frozen - The comparison between the output compare register TIMX_CCR1 and the counter TIMX_CNT has no effect on the outputs.</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>011: Toggle - OC1REF toggles when TIMX_CNT=TIMX_CCR1.</td>
</tr>
<tr>
<td></td>
<td>100: Force inactive level - OC1REF is forced low.</td>
</tr>
<tr>
<td></td>
<td>101: Force active level - OC1REF is forced high.</td>
</tr>
<tr>
<td></td>
<td>110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMX_CNT&lt;TIMX_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF=’0’) as long as TIMX_CNT&gt; TIMX_CCR1 else active (OC1REF=’1’).</td>
</tr>
<tr>
<td></td>
<td>111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMX_CNT&lt;TIMX_CCR1 else active. In downcounting, channel 1 is active as long as TIMX_CNT&gt; TIMX_CCR1 else inactive.</td>
</tr>
<tr>
<td>3</td>
<td><strong>OC1PE</strong>: Output Compare 1 preload enable</td>
</tr>
<tr>
<td></td>
<td>0: Preload register on TIMX_CCR1 disabled. TIMX_CCR1 can be written at anytime, the new value is taken in account immediately.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td><strong>OC1FE</strong>: Output Compare 1 fast enable</td>
</tr>
<tr>
<td></td>
<td>This bit is used to accelerate the effect of an event on the trigger in input on the CC output.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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 of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.</td>
</tr>
</tbody>
</table>

Note: 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: In PWM mode 1 or 2, the OCR1F 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: 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).

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 of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.
Bits 1:0 **CC1S[1:0]**: 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_CCRER).

### 17.6.8 TIMx capture/compare enable register (TIMx_CCRER)(x = 16 to 17)

**Address offset**: 0x20

**Reset value**: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

**Bits 15:4** Reserved, must be kept at reset value.

**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

**Condition: CC1 channel configured as output**

- **0**: OC1 active high
- **1**: OC1 active low

**Condition: CC1 channel configured as input**

The CC1NP/CC1P bits select the polarity of TI1FP1 and TI2FP1 for capture operation.

- **00**: Non-inverted/rising edge: circuit is sensitive to TI1FP1’s rising edge TI1FP1 is not inverted.
- **01**: Inverted/falling edge: circuit is sensitive to TI1FP1’s falling edge, TI1FP1 is inverted.
- **10**: Reserved, do not use this configuration.
- **11**: Non-inverted/both edges: circuit is sensitive to both the rising and falling edges of TI1FP1, TI1FP1 is not 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

**Condition: 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.

**Condition: 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 57. Output control bits for complementary OCx and OCxN channels with break feature

<table>
<thead>
<tr>
<th>Control bits</th>
<th>MOE bit</th>
<th>OSSi bit</th>
<th>OSSR bit</th>
<th>CCxE bit</th>
<th>CCxNE bit</th>
<th>OCx output state</th>
<th>OCxN output state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0</td>
<td>0 0 1</td>
<td>0 1 0</td>
<td>0 1 1</td>
<td>1 0 0</td>
<td>1 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td></td>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
<td>1 0</td>
<td>0 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
17.6.9  TIMx counter (TIMx_CNT)(x = 16 to 17)
Address offset: 0x24
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>CNT[15:0]: Counter value</th>
</tr>
</thead>
</table>

17.6.10 TIMx prescaler (TIMx_PSC)(x = 16 to 17)
Address offset: 0x28
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>PSC[15:0]: Prescaler value</th>
</tr>
</thead>
</table>

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”).

17.6.11 TIMx auto-reload register (TIMx_ARR)(x = 16 to 17)
Address offset: 0x2C
Reset value: 0xFFFF

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>ARR[15:0]: Auto-reload value</th>
</tr>
</thead>
</table>

ARR is the value to be loaded in the actual auto-reload register.
Refer to the Section 16.3.1: Time-base unit on page 381 for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.
17.6.12  TIMx repetition counter register (TIMx_RCR)(x = 16 to 17)

Address offset: 0x30
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Bits 15:8  Reserved, must be kept at reset value.

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 reloading 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.

17.6.13  TIMx capture/compare register 1 (TIMx_CCR1)(x = 16 to 17)

Address offset: 0x34
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CCR1[15:0]</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 15:0  CCR1[15:0]: Capture/Compare 1 value

**Condition: 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.

**Condition: channel CC1 is configured as input**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

17.6.14  TIMx break and dead-time register (TIMx_BDTR)(x = 16 to 17)

Address offset: 0x44
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOE</td>
<td>AOE</td>
<td>BKP</td>
<td>BKE</td>
<td>OSSR</td>
<td>OSSI</td>
<td>LOCK[1:0]</td>
<td>DTG[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
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 17.5.9: TIM15 capture/compare enable register (TIM15_CCER) on page 441).

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 active)
Note: This bit cannot 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 cannot be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).
Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 12 BKE: Break enable
0: Break inputs (BRK and CCS clock failure event) disabled
1: Break inputs (BRK and CCS clock failure event) enabled
Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).
Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

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 17.5.9: TIM15 capture/compare enable register (TIM15_CCER) on page 441).
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 cannot 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 17.5.9: TIM15 capture/compare enable register (TIM15_CCER) on page 441).

- **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 setup

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 tDts with tDtg=tDTS
- **DTG[7:5]=10x** => DT=(64+DTG[5:0])x tDts with Tdsg=2xTdTS
- **DTG[7:5]=110** => DT=(32+DTG[4:0])x tDts with Tdsg=8xTdTS
- **DTG[7:5]=111** => DT=(32+DTG[4:0])x tDts with Tdsg=16xTdTS

Example if T\text{DTS}=125\text{ns} (8\text{MHz}), dead-time possible values are:

- 0 to 15875 ns by 125 ns steps,
- 16 µs to 31750 ns by 250 ns steps,
- 32 µs to 63 µs by 1 µs steps,
- 64 µs to 126 µs by 2 µs 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).

### 17.6.15 TIMx DMA control register (TIMx_DCR)(x = 16 to 17)

**Address offset**: 0x48

**Reset value**: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RW</td>
</tr>
</tbody>
</table>

**Bits 15:13**: Reserved, must be kept at reset value.
17.6.16 TIMx DMA address for full transfer (TIMx_DMAR)(x = 16 to 17)

Address offset: 0x4C
Reset value: 0x0000

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.

Bits 7:5 Reserved, must be kept at reset value.

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
- ...

Example: Let us consider the following transfer: DBL = 7 transfers and DBA = TIMx_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx_CR1 address.

Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers (x = 2, 3, 4) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:
1. Configure the corresponding DMA channel as follows:
   – DMA channel peripheral address is the DMAR register address
   – DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
   – Number of data to transfer = 3 (See note below).
   – Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:
   DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

Note: This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let us take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.

### 17.6.17 TIM16/TIM17 register map

TIM16 and TIM17 registers are mapped as 16-bit addressable registers as described in the table below:

| 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   | TIM16_CR1 and TIM17_CR1   | CKD|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |                           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value               | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x04   | TIM16_CR2 and TIM17_CR2   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |                           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value               | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C   | TIM16_DIER and TIM17_DIER |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |                           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value               | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x10   | TIM16_SR and TIM17_SR     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |                           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value               | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x14   | TIM16_EGR and TIM17_EGR   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |                           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
Refer to Section 2.2 on page 37 for the register boundary addresses.
18 Infrared interface (IRTIM)

An infrared interface (IRTIM) for remote control is available on the device. It can be used with an infrared LED to perform remote control functions.

It uses internal connections with TIM16 as shown in Figure 193.

To generate the infrared remote control signals, the IR interface must be enabled and TIM16 channel 1 (TIM16_OC1) must be properly configured to generate correct waveforms.

The infrared receiver can be implemented easily through a basic input capture mode.

Figure 193. IRTIM internal hardware connections

All standard IR pulse modulation modes can be obtained by programming the two timer output compare channels.

is used to generate the high frequency carrier signal, while TIM16 generates the modulation envelope.

The infrared function is output on the IR_OUT pin. The activation of this function is done through the GPIOx_AFRx register by enabling the related alternate function bit.

The high sink LED driver capability (only available on the PB9 pin) can be activated through the PB9_FMP bit in the SYSCFG_CFRG1 register and used to sink the high current needed to directly control an infrared LED.

For code example refer to the Appendix section A.9.1: TIM16 and TIM17 configuration.
19 Independent watchdog (IWDG)

19.1 Introduction

The devices feature an embedded watchdog peripheral that offers a combination of high safety level, timing accuracy and flexibility of use. The Independent watchdog peripheral detects and solves malfunctions due to software failure, and triggers system reset when the counter reaches a given timeout value.

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails.

The IWDG is best suited for applications that require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. For further information on the window watchdog, refer to Section 20: System window watchdog (WWDG).

19.2 IWDG main features

- Free-running downcounter
- Clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Conditional reset
  - Reset (if watchdog activated) when the downcounter value becomes lower than 0x000
  - Reset (if watchdog activated) if the downcounter is reloaded outside the window

19.3 IWDG functional description

19.3.1 IWDG block diagram

Figure 194 shows the functional blocks of the independent watchdog module.

Figure 194. Independent watchdog block diagram

1. The register interface is located in the CORE voltage domain. The watchdog function is located in the VDD voltage domain, still functional in Stop and Standby modes.
When the independent watchdog is started by writing the value 0x0000 CCCC in the *IWDG 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 0x0000 AAAA is written in the *IWDG key register (IWDG_KR)*, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.

Once running, the IWDG cannot be stopped.

### 19.3.2 Window option

The IWDG can also work as a window watchdog by setting the appropriate window in the *IWDG window register (IWDG_WINR)*.

If the reload operation is performed while the counter is greater than the value stored in the *IWDG window register (IWDG_WINR)*, then a reset is provided.

The default value of the *IWDG window register (IWDG_WINR)* is 0x0000 0FFF, so if it is not updated, the window option is disabled.

As soon as the window value is changed, a reload operation is performed in order to reset the downcounter to the *IWDG reload register (IWDG_RLR)* value and ease the cycle number calculation to generate the next reload.

**Configuring the IWDG when the window option is enabled**

1. Enable the IWDG by writing 0x0000 CCCC in the *IWDG key register (IWDG_KR)*.
2. Enable register access by writing 0x0000 5555 in the *IWDG key register (IWDG_KR)*.
3. Write the IWDG prescaler by programming *IWDG prescaler register (IWDG_PR)* from 0 to 7.
4. Write the *IWDG reload register (IWDG_RLR)*.
5. Wait for the registers to be updated (IWDG_SR = 0x0000 0000).
6. Write to the *IWDG window register (IWDG_WINR)*. This automatically refreshes the counter value in the *IWDG reload register (IWDG_RLR)*.

*Note:* Writing the window value allows the counter value to be refreshed by the RLR when IWDG status register (IWDG_SR) is set to 0x0000 0000.

For code example refer to the Appendix section A.12.2: IWDG configuration with window.

**Configuring the IWDG when the window option is disabled**

When the window option it is not used, the IWDG can be configured as follows:

1. Enable the IWDG by writing 0x0000 CCCC in the *IWDG key register (IWDG_KR)*.
2. Enable register access by writing 0x0000 5555 in the *IWDG key register (IWDG_KR)*.
3. Write the prescaler by programming the *IWDG prescaler register (IWDG_PR)* from 0 to 7.
4. Write the *IWDG reload register (IWDG_RLR)*.
5. Wait for the registers to be updated (IWDG_SR = 0x0000 0000).
6. Refresh the counter value with IWDG_RLR (IWDG_KR = 0x0000 AAAA).

For code example refer to the Appendix section A.12.1: IWDG configuration.
19.3.3 **Hardware watchdog**

If the “Hardware watchdog” feature is enabled through the device option bits, the watchdog is automatically enabled at power-on, and generates a reset unless the **IWDG key register (IWDG_KR)** is written by the software before the counter reaches end of count or if the downcounter is reloaded inside the window.

19.3.4 **Register access protection**

Write access to **IWDG prescaler register (IWDG_PR)**, **IWDG reload register (IWDG_RLR)** and **IWDG window register (IWDG_WINR)** is protected. To modify them, the user must first write the code 0x0000 5555 in the **IWDG key register (IWDG_KR)**. A write access to this register with a different value breaks the sequence and register access is protected again. This is the case of the reload operation (writing 0x0000 AAAAA).

A status register is available to indicate that an update of the prescaler or of the downcounter reload value or of the window value is ongoing.

For code example refer to the Appendix section **A.12.1: IWDG configuration**.

19.3.5 **Debug mode**

When the device enters Debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on the configuration of the corresponding bit in DBGMCU freeze register.
19.4 **IWDG registers**

Refer to *Section 1.2 on page 33* for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

19.4.1 **IWDG key register (IWDG KR)**

Address offset: 0x00

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

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEY[15:0]**

- Bits 31:16: Reserved, must be kept at reset value.
- Bits 15:0 **KEY[15:0]**: Key value (write only, read 0x0000)
  - These bits must be written by software at regular intervals with the key value 0xAAAA, otherwise the watchdog generates a reset when the counter reaches 0.
  - Writing the key value 0x5555 to enable access to the IWDG_PR, IWDG_RLR and IWDG_WINR registers (see *Section 19.3.4: Register access protection*).
  - Writing the key value 0xC000 starts the watchdog (except if the hardware watchdog option is selected).
19.4.2 IWDG prescaler register (IWDG_PR)

Address offset: 0x04
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PR[2:0]</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:3 Reserved, must be kept at reset value.

Bits 2:0 PR[2:0]: Prescaler divider

These bits are write access protected see Section 19.3.4: Register access protection. They are written by software to select the prescaler divider feeding the counter clock. PVU bit of the IWDG status register (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 status register (IWDG_SR) is reset.
### 19.4.3 IWDG 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 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

- **Bits 31:12** Reserved, must be kept at reset value.
- **Bits 11:0** **RL[11:0]:** Watchdog counter reload value
  
  These bits are write access protected see [Register access protection](#). They are written by software to define the value to be loaded in the watchdog counter each time the value 0xAAAA is written in the [IWDG key register (IWDG_KR)](#). The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to the datasheet for the timeout information.

  The RVU bit in the [IWDG status register (IWDG_SR)](#) must be reset to be able to change the reload value.

  **Note:** Reading this register returns the reload value from the V\textsubscript{DD} voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on it. For this reason the value read from this register is valid only when the RVU bit in the IWDG status register (IWDG_SR) is reset.
19.4.4  IWDG status register (IWDG_SR)

Address offset: 0x0C
Reset value: 0x0000 0000 (not reset by Standby mode)

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>Bit 30</th>
<th>Bit 29</th>
<th>Bit 28</th>
<th>Bit 27</th>
<th>Bit 26</th>
<th>Bit 25</th>
<th>Bit 24</th>
<th>Bit 23</th>
<th>Bit 22</th>
<th>Bit 21</th>
<th>Bit 20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bit 16</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WVU</td>
<td>RVU</td>
</tr>
</tbody>
</table>

Bits 31:3  Reserved, must be kept at reset value.

Bit 2  WVU: Watchdog counter window value update
This bit is set by hardware to indicate that an update of the window 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 five RC 40 kHz cycles).
Window value can be updated only when WVU bit is reset.

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 five 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 five RC 40 kHz cycles).
Prescaler value can be updated only when PVU bit is reset.

Note:  If several reload, prescaler, or window values are used by the application, it is mandatory to wait until RVU bit is reset before changing the reload value, to wait until PVU bit is reset before changing the prescaler value, and to wait until WVU bit is reset before changing the window value. However, after updating the prescaler and/or the reload/window value it is not necessary to wait until RVU or PVU or WVU is reset before continuing code execution except in case of low-power mode entry.
19.4.5  IWDG window register (IWDG_WINR)

Address offset: 0x10
Reset value: 0x0000 0FFF (reset by Standby mode)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 31:12  Reserved, must be kept at reset value.

Bits 11:0  **WIN[11:0]**: Watchdog counter window value

These bits are write access protected, see Section 19.3.4, they contain the high limit of the window value to be compared with the downcounter.

To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x0

The WVU bit in the **IWDG status register (IWDG_SR)** 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 valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the WVU bit in the **IWDG status register (IWDG_SR)** is reset.
### 19.4.6 IWDG register map

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

**Table 59. IWDG register map and reset values**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>IWDG_KR</td>
<td>0x04</td>
<td>IWDG PR</td>
<td>0x08</td>
<td>IWDG RLR</td>
<td>0x0C</td>
<td>IWDG SR</td>
</tr>
<tr>
<td>Reset value</td>
<td></td>
<td>Reset value</td>
<td></td>
<td>Reset value</td>
<td></td>
<td>Reset value</td>
<td></td>
</tr>
<tr>
<td>0000000000000000</td>
<td></td>
<td>0000000000000000</td>
<td></td>
<td>1111111111111111</td>
<td></td>
<td>0000000000000000</td>
<td></td>
</tr>
</tbody>
</table>

Refer to *Section 2.2 on page 37* for the register boundary addresses.
System window watchdog (WWDG)

20.1 Introduction

The system window watchdog (WWDG) 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 down-counter before the T6 bit is cleared. An MCU reset is also generated if the 7-bit down-counter value (in the control register) is refreshed before the down-counter reaches the window register value. This implies that the counter must be refreshed in a limited window.

The WWDG clock is prescaled from the APB clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

The WWDG is best suited for applications requiring the watchdog to react within an accurate timing window.

20.2 WWDG main features

- Programmable free-running down-counter
- Conditional reset
  - Reset (if watchdog activated) when the down-counter value becomes lower than 0x40
  - Reset (if watchdog activated) if the down-counter is reloaded outside the window (see Figure 196)
- Early wake-up interrupt (EWI): triggered (if enabled and the watchdog activated) when the down-counter is equal to 0x40

20.3 WWDG functional description

If the watchdog is activated (the WDGA bit is set in the WWDG_CR register), and when the 7-bit down-counter (T[6:0] bits) is decremented 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.

The application program must write in the WWDG_CR register at regular intervals during normal operation to prevent an MCU reset. This operation can take place only when the counter value is lower than or equal to the window register value, and higher than 0x3F. The value to be stored in the WWDG_CR register must be between 0xFF and 0xC0.

Refer to Figure 195 for the WWDG block diagram.
20.3.1 WWDG block diagram

Figure 195. Watchdog block diagram

20.3.2 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.

20.3.3 Controlling the down-counter

This down-counter is free-running, counting 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 that represent 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 196). The WWDG configuration register (WWDG_CFR) contains the high limit of the window: to prevent a reset, the down-counter must be reloaded when its value is lower than or equal to the window register value, and greater than 0x3F. Figure 196 describes the window watchdog process.

Note: The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).

20.3.4 How to program the watchdog timeout

Use the formula in Figure 196 to calculate the WWDG timeout.

Warning: When writing to the WWDG_CR register, always write 1 in the T6 bit to avoid generating an immediate reset.
The formula to calculate the timeout value is given by:

\[
t_{\text{WWDG}} = t_{\text{PCLK}} \times 4096 \times 2^{\text{WDGTB}[1:0]} \times (T[5:0] + 1) \quad (\text{ms})
\]

where:
- \( t_{\text{WWDG}} \): WWDG timeout
- \( t_{\text{PCLK}} \): APB clock period measured in ms
- 4096: Value corresponding to internal divider

As an example, if APB frequency is 48 MHz, WDGTB[1:0] is set to 3 and T[5:0] is set to 63:

\[
t_{\text{WWDG}} = \left( \frac{1}{48000} \right) \times 4096 \times 2^3 \times (63 + 1) = 43.69 \text{ ms}
\]

Refer to the datasheet for the minimum and maximum values of \( t_{\text{WWDG}} \).

### 20.3.5 Debug mode

When the device enters debug mode (processor halted), the WWDG counter either continues to work normally or stops, depending on the configuration bit in DBG module. For more details, refer to Section 26.9.2: Debug support for timers, watchdog and I2C.
20.4 **WWDG interrupts**

The early wake-up interrupt (EWI) can be used if specific safety operations or data logging must be performed before the reset is generated. To enable the early wake-up interrupt, the application must:

- Write EWIF bit of WWDG_SR register to 0, to clear unwanted pending interrupt
- Write EWI bit of WWDG_CFR register to 1, to enable interrupt

When the down-counter reaches the value 0x40, a watchdog interrupt is generated, and the corresponding interrupt service routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case the corresponding ISR must reload the WWDG counter to avoid the WWDG reset, then trigger the required actions.

The watchdog interrupt is cleared by writing '0' to the EWIF bit in the WWDG_SR register.

*Note:* When the watchdog interrupt cannot be served (for example due to a system lock in a higher priority task), the WWDG reset is eventually generated.

20.5 **WWDG registers**

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by halfwords (16-bit) or words (32-bit).

20.5.1 **WWDG control register (WWDG_CR)**

Address offset: 0x000

Reset value: 0x0000 007F

<table>
<thead>
<tr>
<th>Bit 31:8</th>
<th>Bit 7:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs</td>
<td>rs</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>rs</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:8 Reserved, must be kept at reset value.

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, decremented every $(4096 \times 2^{WDGTB[1:0]})$ PCLK cycles. A reset is produced when it is decremented from 0x40 to 0x3F (T6 becomes cleared).
## 20.5.2 WWDG configuration register (WWDG_CFR)

Address offset: 0x004  
Reset value: 0x0000 007F

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

Address offset: 0x008  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

- **Bits 31:10**: Reserved, must be kept at reset value.
- **Bit 9**: **EWI**: Early wake-up interrupt enable  
  Set by software and cleared by hardware after a reset. When set, an interrupt occurs whenever the counter reaches the value 0x40.

- **Bits 8:7**: **WDGTB[1:0]**: Timer base  
  The time base of the prescaler can be modified as follows:  
  - 00: CK counter clock (PCLK div 4096) div 1  
  - 01: CK counter clock (PCLK div 4096) div 2  
  - 10: CK counter clock (PCLK div 4096) div 4  
  - 11: CK counter clock (PCLK div 4096) div 8

- **Bits 6:0**: **W[6:0]**: 7-bit window value  
  These bits contain the window value to be compared with the down-counter.

## 20.5.3 WWDG status register (WWDG_SR)

Address offset: 0x008  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

Address offset: 0x008  
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

- **Bits 31:1**: Reserved, must be kept at reset value.
- **Bit 0**: **EWIF**: Early wake-up interrupt flag  
  This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. Writing 1 has no effect. This bit is also set if the interrupt is not enabled.
20.5.4 **WWDG register map**

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

**Table 60. WWDG register map and reset values**

| Offset | Register name | 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  | WWDG_CR       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x004  | WWDG_CFR      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x008  | WWDG_SR       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Refer to *Section 2.2 on page 37* for the register boundary addresses.
21 Real-time clock (RTC)

21.1 Introduction

The RTC provides an automatic wake-up to manage all low-power modes.

The real-time clock (RTC) is an independent BCD timer/counter. The RTC provides a time-of-day clock/calendar with programmable alarm interrupt.

The RTC includes also a periodic programmable wake-up flag with interrupt capability.

Two 32-bit registers contain the seconds, minutes, hours (12- or 24-hour format), day (day of week), date (day of month), month, and year, expressed in binary coded decimal format (BCD). The sub-seconds value is also available in binary format.

Compensations for 28-, 29- (leap year), 30-, and 31-day months are performed automatically. Daylight saving time compensation can also be performed.

Additional 32-bit registers contain the programmable alarm subseconds, seconds, minutes, hours, day, and date.

A digital calibration feature is available to compensate for any deviation in crystal oscillator accuracy.

After RTC domain reset, all RTC registers are protected against possible parasitic write accesses.

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low-power mode or under reset).
21.2 RTC main features

The RTC unit main features are the following (see Figure 197: RTC block diagram in STM32F030x4/6, STM32F070x6 and STM32F030x8 devices and Figure 198: RTC block diagram for STM32F070xB and STM32F030xC devices):

- Calendar with subseconds, seconds, minutes, hours (12 or 24 format), day (day of week), date (day of month), month, and year.
- Daylight saving compensation programmable by software.
- Programmable alarm with interrupt function. The alarm can be triggered by any combination of the calendar fields.
- Automatic wake-up unit generating a periodic flag that triggers an automatic wake-up interrupt.
- Reference clock detection: a more precise second source clock (50 or 60 Hz) can be used to enhance the calendar precision.
- Accurate synchronization with an external clock using the subsecond shift feature.
- Digital calibration circuit (periodic counter correction): 0.95 ppm accuracy, obtained in a calibration window of several seconds
- Time-stamp function for event saving
- Tamper detection event with configurable filter and internal pull-up
- Maskable interrupts/events:
  - Alarm A
  - Wake-up interrupt
  - Time-stamp
  - Tamper detection

21.3 RTC implementation

<table>
<thead>
<tr>
<th>RTC Features</th>
<th>STM32F030x4, STM32F030x6, STM32F070x6, STM32F030x8</th>
<th>STM32F070xB STM32F030xC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic wake-up timer</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>RTC_TAMP1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RTC_TAMP2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RTC_TAMP3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Alarm A</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

1. X = supported, '-' = not supported.
21.4 RTC functional description

21.4.1 RTC block diagram

Figure 197. RTC block diagram in STM32F030x4/6, STM32F070x6 and STM32F030x8 devices
The RTC includes:

- One alarm
- Two tamper events from I/Os
- One timestamp event from I/O
- Tamper event detection can generate a timestamp event
- Output functions: RTC_OUT which selects one of the following two outputs:
  - RTC_CALIB: 512 Hz or 1Hz clock output (with an LSE frequency of 32.768 kHz). This output is enabled by setting the COE bit in the RTC_CR register.
  - RTC_ALARM: Alarm A. This output is selected by configuring the OSEL[1:0] bits in the RTC_CR register.
- Input functions:
  - RTC_TS: timestamp event
  - RTC_TAMP1: tamper1 event detection
  - RTC_TAMP2: tamper2 event detection
  - RTC_REFIN: 50 or 60 Hz reference clock input
21.4.2 **GPIOs controlled by the RTC**

RTC_OUT, RTC_TS and RTC_TAMP1 are mapped on the same pin (PC13). PC13 pin configuration is controlled by the RTC, whatever the PC13 GPIO configuration, except for the RTC_ALARM output open-drain mode. In this particular case, the GPIO must be configured as input. The RTC functions mapped on PC13 are available in all low-power modes and in VBAT mode.

The selection of the RTC_ALARM output is performed through the RTC_TAFCR register as follows: the PC13VALUE bit is used to select whether the RTC_ALARM output is configured in push-pull or open drain mode.

When PC13 is not used as RTC function, it can be forced in output push-pull mode by setting the PC13MODE bit in the RTC_TAFCR. The output data value is then given by the PC13VALUE bit. In this case, PC13 output push-pull state and data are preserved in Standby mode.

The output mechanism follows the priority order shown in Table 62.

When PC14 and PC15 are not used as LSE oscillator, they can be forced in output push-pull mode by setting the PC14MODE and PC15MODE bits in the RTC_TAFCR register respectively. The output data values are then given by PC14VALUE and PC15VALUE. In this case, the PC14 and PC15 output push-pull states and data values are preserved in Standby mode.

The output mechanism follows the priority order shown in Table 63 and Table 64.

<table>
<thead>
<tr>
<th>Pin configuration and function</th>
<th>RTC_ALARM output enabled</th>
<th>RTC_CALIB output enabled</th>
<th>RTC_TAMP1 input enabled</th>
<th>RTC_TS input enabled</th>
<th>PC13MODE bit</th>
<th>PC13VALUE bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_ALARM output OD</td>
<td>1</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>0</td>
</tr>
<tr>
<td>RTC_ALARM output PP</td>
<td>1</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>1</td>
</tr>
<tr>
<td>RTC_CALIB output PP</td>
<td>0</td>
<td>1</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>RTC_TAMP1 input floating</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>RTC_TS and RTC_TAMP1 input floating</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>RTC_TS input floating</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>Output PP forced</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PC13 output data value</td>
</tr>
<tr>
<td>Wake-up pin or Standard GPIO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Don’t care</td>
</tr>
</tbody>
</table>

1. OD: open drain; PP: push-pull.
21.4.3 Clock and prescalers

The RTC clock source (RTCLK) is selected through the clock controller among the LSE clock, the LSI oscillator clock, and the HSE clock. For more information on the RTC clock source configuration, refer to .

A programmable prescaler stage generates a 1 Hz clock which is used to update the calendar. To minimize power consumption, the prescaler is split into 2 programmable prescalers (see Figure 197: RTC block diagram in STM32F030x4/6, STM32F070x6 and STM32F030x8 devices):

- A 7-bit asynchronous prescaler configured through the PREDIV_A bits of the RTC_PRER register.
- A 15-bit synchronous prescaler configured through the PREDIV_S bits of the RTC_PRER register.

**Note:** When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize consumption.

The asynchronous prescaler division factor is set to 128, and the synchronous division factor to 256, to obtain an internal clock frequency of 1 Hz (ck_spre) with an LSE frequency of 32.768 kHz.

The minimum division factor is 1 and the maximum division factor is $2^{22}$.

This corresponds to a maximum input frequency of around 4 MHz.

### Table 63. LSE pin PC14 configuration (1)

<table>
<thead>
<tr>
<th>Pin configuration and function</th>
<th>LSEON bit in RCC_BDCR register</th>
<th>LSEBYP bit in RCC_BDCR register</th>
<th>PC14MODE bit</th>
<th>PC14VALUE bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSE oscillator</td>
<td>1</td>
<td>0</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>LSE bypass</td>
<td>1</td>
<td>1</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>Output PP forced</td>
<td>0</td>
<td>Don’t care</td>
<td>1</td>
<td>PC14 output data value</td>
</tr>
<tr>
<td>Standard GPIO</td>
<td>0</td>
<td>Don’t care</td>
<td>0</td>
<td>Don’t care</td>
</tr>
</tbody>
</table>

1. OD: open drain; PP: push-pull.

### Table 64. LSE pin PC15 configuration (1)

<table>
<thead>
<tr>
<th>Pin configuration and function</th>
<th>LSEON bit in RCC_BDCR register</th>
<th>LSEBYP bit in RCC_BDCR register</th>
<th>PC15MODE bit</th>
<th>PC15VALUE bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSE oscillator</td>
<td>1</td>
<td>0</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>Output PP forced</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PC15 output data value</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Don’t care</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard GPIO</td>
<td>0</td>
<td>Don’t care</td>
<td>0</td>
<td>Don’t care</td>
</tr>
</tbody>
</table>

1. OD: open drain; PP: push-pull.
The ck_apre clock is used to clock the binary RTC_SSR subseconds downcounter. When it reaches 0, RTC_SSR is reloaded with the content of PREDIV_S.

The ck_spre clock can be used either to update the calendar or as timebase for the 16-bit wake-up auto-reload timer. To obtain short timeout periods, the 16-bit wake-up auto-reload timer can also run with the RTCCLK divided by the programmable 4-bit asynchronous prescaler (see Section 21.4.6: Periodic auto-wake-up for details).

21.4.4 Real-time clock and calendar

The RTC calendar time and date registers are accessed through shadow registers which are synchronized with PCLK (APB clock). They can also be accessed directly in order to avoid waiting for the synchronization duration.

- RTC_SSR for the subseconds
- RTC_TR for the time
- RTC_DR for the date

Every RTCCLK period, the current calendar value is copied into the shadow registers, and the RSF bit of RTC_ISR register is set (see Section 21.7.4: RTC initialization and status register (RTC_ISR)). The copy is not performed in Stop and Standby mode. When exiting these modes, the shadow registers are updated after up to 1 RTCCLK period.

When the application reads the calendar registers, it accesses the content of the shadow registers. It is possible to make a direct access to the calendar registers by setting the BYPSHAD control bit in the RTC_CR register. By default, this bit is cleared, and the user accesses the shadow registers.

When reading the RTC_SSR, RTC_TR or RTC_DR registers in BYPSHAD=0 mode, the frequency of the APB clock (fAPB) must be at least 7 times the frequency of the RTC clock (fRTCCLK).

The shadow registers are reset by system reset.

21.4.5 Programmable alarm

The RTC unit provides programmable alarm: Alarm A.

The programmable alarm function is enabled through the ALRAE bit in the RTC_CR register. The ALRAF is set to 1 if the calendar subseconds, seconds, minutes, hours, date or day match the values programmed in the alarm registers RTC_ALRMASSR and RTC_ALRMA. Each calendar field can be independently selected through the MSKx bits of the RTC_ALRMA register, and through the MASKSSx bits of the RTC_ALRMASSR register. The alarm interrupt is enabled through the ALRAIE bit in the RTC_CR register.
Caution: If the seconds field is selected (MSK1 bit reset in RTC_ALRMAR), the synchronous prescaler division factor set in the RTC_PRER register must be at least 3 to ensure correct behavior.

Alarm A (if enabled by bits OSEL[1:0] in RTC_CR register) can be routed to the RTC_ALARM output. RTC_ALARM output polarity can be configured through bit POL the RTC_CR register.

21.4.6 Periodic auto-wake-up

The periodic wake-up flag is generated by a 16-bit programmable auto-reload down-counter. The wake-up timer range can be extended to 17 bits.

The wake-up function is enabled through the WUTE bit in the RTC_CR register.

The wake-up timer clock input can be:

- RTC clock (RTCCLK) divided by 2, 4, 8, or 16.
  When RTCCLK is LSE(32.768 kHz), this allows to configure the wake-up interrupt period from 122 µs to 32 s, with a resolution down to 61 µs.
- ck_spre (usually 1 Hz internal clock)
  When ck_spre frequency is 1Hz, this allows to achieve a wake-up time from 1 s to around 36 hours with one-second resolution. This large programmable time range is divided in 2 parts:
  - from 1s to 18 hours when WUCKSEL [2:1] = 10
  - and from around 18h to 36h when WUCKSEL[2:1] = 11. In this last case 216 is added to the 16-bit counter current value. When the initialization sequence is complete (see Programming the wake-up timer on page 493), the timer starts counting down. When the wake-up function is enabled, the down-counting remains active in low-power modes. In addition, when it reaches 0, the WUTF flag is set in the RTC_ISR register, and the wake-up counter is automatically reloaded with its reload value (RTC_WUTR register value).

The WUTF flag must then be cleared by software.

When the periodic wake-up interrupt is enabled by setting the WUTIE bit in the RTC_CR register, it can exit the device from low-power modes.

The periodic wake-up flag can be routed to the RTC_ALARM output provided it has been enabled through bits OSEL[1:0] of RTC_CR register. RTC_ALARM output polarity can be configured through the POL bit in the RTC_CR register.

System reset, as well as low-power modes (Sleep, Stop and Standby) have no influence on the wake-up timer.

21.4.7 RTC initialization and configuration

RTC register access

The RTC registers are 32-bit registers. The APB interface introduces 2 wait-states in RTC register accesses except on read accesses to calendar shadow registers when BYPSHAD=0.
RTC register write protection

After system reset, the RTC registers are protected against parasitic write access by clearing the DBP bit in the PWR_CR register (refer to the power control section). DBP bit must be set in order to enable RTC registers write access.

After RTC domain reset, all the RTC registers are write-protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.

The following steps are required to unlock the write protection on all the RTC registers except for RTC_TAFCR and RTC_ISR[13:8].

1. Write ‘0xCA’ into the RTC_WPR register.
2. Write ‘0x53’ into the RTC_WPR register.

Writing a wrong key reactivates the write protection.

The protection mechanism is not affected by system reset.

Calendar initialization and configuration

To program the initial time and date calendar values, including the time format and the prescaler configuration, the following sequence is required:

1. Set INIT bit to 1 in the RTC_ISR register to enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated.
2. Poll INITF bit of in the RTC_ISR register. The initialization phase mode is entered when INITF is set to 1. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. To generate a 1 Hz clock for the calendar counter, program both the prescaler factors in RTC_PRER register.
4. Load the initial time and date values in the shadow registers (RTC_TR and RTC_DR), and configure the time format (12 or 24 hours) through the FMT bit in the RTC_CR register.
5. Exit the initialization mode by clearing the INIT bit. The actual calendar counter value is then automatically loaded and the counting restarts after 4 RTCCLK clock cycles.

When the initialization sequence is complete, the calendar starts counting.

Note: After a system reset, the application can read the INITS flag in the RTC_ISR register to check if the calendar has been initialized or not. If this flag equals 0, the calendar has not been initialized since the year field is set at its RTC domain reset default value (0x00).

To read the calendar after initialization, the software must first check that the RSF flag is set in the RTC_ISR register.

For code example refer to the Appendix section A.13.1: RTC calendar configuration.

Daylight saving time

The daylight saving time management is performed through bits SUB1H, ADD1H, and BKP of the RTC_CR register.

Using SUB1H or ADD1H, the software can subtract or add one hour to the calendar in one single operation without going through the initialization procedure.

In addition, the software can use the BKP bit to memorize this operation.
Programming the alarm

A similar procedure must be followed to program or update the programmable alarms.

1. Clear ALRAE in RTC_CR to disable Alarm A.
2. Program the Alarm A registers (RTC_ALRMASSR/RTC_ALRMAR).
3. Set ALRAE in the RTC_CR register to enable Alarm A again.

*Note:* Each change of the RTC_CR register is taken into account after around 2 RTCCLK clock cycles due to clock synchronization.

For code example refer to the Appendix section A.13.2: RTC alarm configuration.

Programming the wake-up timer

The following sequence is required to configure or change the wake-up timer auto-reload value (WUT[15:0] in RTC_WUTR):

1. Clear WUTE in RTC_CR to disable the wake-up timer.
2. Poll WUTWF until it is set in RTC_ISR to make sure the access to wake-up auto-reload counter and to WUCKSEL[2:0] bits is allowed. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. Program the wake-up auto-reload value WUT[15:0], and the wake-up clock selection (WUCKSEL[2:0] bits in RTC_CR). Set WUTE in RTC_CR to enable the timer again. The wake-up timer restarts down-counting. The WUTWF bit is cleared up to 2 RTCCLK clock cycles after WUTE is cleared, due to clock synchronization.

For code example refer to the Appendix section A.13.3: RTC WUT configuration.

21.4.8 Reading the calendar

When BYPSHAD control bit is cleared in the RTC_CR register

To read the RTC calendar registers (RTC_SSR, RTC_TR and RTC_DR) properly, the APB clock frequency \(f_{PCLK}\) must be equal to or greater than seven times the RTC clock frequency \(f_{RTCCLK}\). This ensures a secure behavior of the synchronization mechanism.

If the APB clock frequency is less than seven times the RTC clock frequency, the software must read the calendar time and date registers twice. If the second read of the RTC_TR gives the same result as the first read, this ensures that the data is correct. Otherwise a third read access must be done. In any case the APB clock frequency must never be lower than the RTC clock frequency.

The RSF bit is set in RTC_ISR register each time the calendar registers are copied into the RTC_SSR, RTC_TR and RTC_DR shadow registers. The copy is performed every RTCCLK cycle. To ensure consistency between the 3 values, reading either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read. In case the software makes read accesses to the calendar in a time interval smaller than 1 RTCCLK period: RSF must be cleared by software after the first calendar read, and then the software must wait until RSF is set before reading again the RTC_SSR, RTC_TR and RTC_DR registers.

After waking up from low-power mode (Stop or Standby), RSF must be cleared by software. The software must then wait until it is set again before reading the RTC_SSR, RTC_TR and RTC_DR registers.

The RSF bit must be cleared after wake-up and not before entering low-power mode.
After a system reset, the software must wait until RSF is set before reading the RTC_SSR, RTC_TR and RTC_DR registers. Indeed, a system reset resets the shadow registers to their default values.

After an initialization (refer to Calendar initialization and configuration on page 492): the software must wait until RSF is set before reading the RTC_SSR, RTC_TR and RTC_DR registers.

After synchronization (refer to Section 21.4.10: RTC synchronization): the software must wait until RSF is set before reading the RTC_SSR, RTC_TR and RTC_DR registers.

For code example refer to the Appendix section A.13.4: RTC read calendar.

When the BYPSHAD control bit is set in the RTC_CR register (bypass shadow registers)

Reading the calendar registers gives the values from the calendar counters directly, thus eliminating the need to wait for the RSF bit to be set. This is especially useful after exiting from low-power modes (STOP or Standby), since the shadow registers are not updated during these modes.

When the BYPSHAD bit is set to 1, the results of the different registers might not be coherent with each other if an RCCLK edge occurs between two read accesses to the registers. Additionally, the value of one of the registers may be incorrect if an RCCLK edge occurs during the read operation. The software must read all the registers twice, and then compare the results to confirm that the data is coherent and correct. Alternatively, the software can just compare the two results of the least-significant calendar register.

Note: While BYPSHAD=1, instructions which read the calendar registers require one extra APB cycle to complete.

21.4.9 Resetting the RTC

The calendar shadow registers (RTC_SSR, RTC_TR and RTC_DR) and some bits of the RTC status register (RTC_ISR) are reset to their default values by all available system reset sources.

On the contrary, the following registers are reset to their default values by a RTC domain reset and are not affected by a system reset: the RTC current calendar registers, the RTC control register (RTC_CR), the prescaler register (RTC_PRER), the RTC calibration register (RTC_CALR), the RTC shift register (RTC_SHIFTR), the RTC timestamp registers (RTC_TSSSR, RTC_TSSR and RTC_TSDR), the RTC tamper and alternate function configuration register (RTC_TAFCR), the wake-up timer register (RTC_WUTR), the Alarm A registers (RTC_ALRMASSR/RTC_ALRMAR).

In addition, when it is clocked by the LSE, the RTC keeps on running under system reset if the reset source is different from the RTC domain reset one (refer to the RTC clock section of the Reset and clock controller for details on the list of RTC clock sources not affected by system reset). When a RTC domain reset occurs, the RTC is stopped and all the RTC registers are set to their reset values.

21.4.10 RTC synchronization

The RTC can be synchronized to a remote clock with a high degree of precision. After reading the sub-second field (RTC_SSR or RTC_TSSSR), a calculation can be made of the precise offset between the times being maintained by the remote clock and the RTC. The
RTC can then be adjusted to eliminate this offset by “shifting” its clock by a fraction of a second using RTC_SHIFTR.

RTC_SSR contains the value of the synchronous prescaler counter. This allows one to calculate the exact time being maintained by the RTC down to a resolution of 1 / (PREDIV_S + 1) seconds. As a consequence, the resolution can be improved by increasing the synchronous prescaler value (PREDIV_S[14:0]). The maximum resolution allowed (30.52 μs with a 32768 Hz clock) is obtained with PREDIV_S set to 0x7FFF.

However, increasing PREDIV_S means that PREDIV_A must be decreased in order to maintain the synchronous prescaler output at 1 Hz. In this way, the frequency of the asynchronous prescaler output increases, which may increase the RTC dynamic consumption.

The RTC can be finely adjusted using the RTC shift control register (RTC_SHIFTR). Writing to RTC_SHIFTR can shift (either delay or advance) the clock by up to a second with a resolution of 1 / (PREDIV_S + 1) seconds. The shift operation consists of adding the SUBFS[14:0] value to the synchronous prescaler counter SS[15:0]: this will delay the clock. If at the same time the ADD1S bit is set, this results in adding one second and at the same time subtracting a fraction of second, so this will advance the clock.

Caution: Before initiating a shift operation, the user must check that SS[15] = 0 in order to ensure that no overflow will occur.

As soon as a shift operation is initiated by a write to the RTC_SHIFTR register, the SHPF flag is set by hardware to indicate that a shift operation is pending. This bit is cleared by hardware as soon as the shift operation has completed.

Caution: This synchronization feature is not compatible with the reference clock detection feature: firmware must not write to RTC_SHIFTR when REFCKON=1.

21.4.11 RTC reference clock detection

The update of the RTC calendar can be synchronized to a reference clock, RTC_REFIN, which is usually the mains frequency (50 or 60 Hz). The precision of the RTC_REFIN reference clock should be higher than the 32.768 kHz LSE clock. When the RTC_REFIN detection is enabled (REFCKON bit of RTC_CR set to 1), the calendar is still clocked by the LSE, and RTC_REFIN is used to compensate for the imprecision of the calendar update frequency (1 Hz).

Each 1 Hz clock edge is compared to the nearest RTC_REFIN clock edge (if one is found within a given time window). In most cases, the two clock edges are properly aligned. When the 1 Hz clock becomes misaligned due to the imprecision of the LSE clock, the RTC shifts the 1 Hz clock a bit so that future 1 Hz clock edges are aligned. Thanks to this mechanism, the calendar becomes as precise as the reference clock.

The RTC detects if the reference clock source is present by using the 256 Hz clock (ck_apre) generated from the 32.768 kHz quartz. The detection is performed during a time window around each of the calendar updates (every 1 s). The window equals 7 ck_apre periods when detecting the first reference clock edge. A smaller window of 3 ck_apre periods is used for subsequent calendar updates.

Each time the reference clock is detected in the window, the synchronous prescaler which outputs the ck_spcre clock is forced to reload. This has no effect when the reference clock and the 1 Hz clock are aligned because the prescaler is being reloaded at the same moment. When the clocks are not aligned, the reload shifts future 1 Hz clock edges a little for them to be aligned with the reference clock.
If the reference clock halts (no reference clock edge occurred during the 3 ck_apre window), the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a large 7 ck_apre period detection window centered on the ck_spre edge.

When the RTC_REFIN detection is enabled, PREDIV_A and PREDIV_S must be set to their default values:

- PREDIV_A = 0x007F
- PREDIV_S = 0x00FF

*Note: RTC_REFIN clock detection is not available in Standby mode.*

### 21.4.12 RTC smooth digital calibration

The RTC frequency can be digitally calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm. The correction of the frequency is performed using series of small adjustments (adding and/or subtracting individual RTCCLK pulses). These adjustments are fairly well distributed so that the RTC is well calibrated even when observed over short durations of time.

The smooth digital calibration is performed during a cycle of about $2^{20}$ RTCCLK pulses, or 32 seconds when the input frequency is 32768 Hz. This cycle is maintained by a 20-bit counter, cal_cnt[19:0], clocked by RTCCLK.

The smooth calibration register (RTC_CALR) specifies the number of RTCCLK clock cycles to be masked during the 32-second cycle:

- Setting the bit CALM[0] to 1 causes exactly one pulse to be masked during the 32-second cycle.
- Setting CALM[1] to 1 causes two additional cycles to be masked
- Setting CALM[2] to 1 causes four additional cycles to be masked
- and so on up to CALM[8] set to 1 which causes 256 clocks to be masked.

*Note: CALM[8:0] (RTC_CALR) specifies the number of RTCCLK pulses to be masked during the 32-second cycle. Setting the bit CALM[0] to ‘1’ causes exactly one pulse to be masked during the 32-second cycle at the moment when cal_cnt[19:0] is 0x80000; CALM[1]=1 causes two other cycles to be masked (when cal_cnt is 0x40000 and 0xC0000); CALM[2]=1 causes four other cycles to be masked (cal_cnt = 0x20000/0x60000/0xA0000/ 0xE0000); and so on up to CALM[8]=1 which causes 256 clocks to be masked (cal_cnt = 0xXX800).*

While CALM allows the RTC frequency to be reduced by up to 487.1 ppm with fine resolution, the bit CALP can be used to increase the frequency by 488.5 ppm. Setting CALP to ‘1’ effectively inserts an extra RTCCLK pulse every $2^{11}$ RTCCLK cycles, which means that 512 clocks are added during every 32-second cycle.

Using CALM together with CALP, an offset ranging from -511 to +512 RTCCLK cycles can be added during the 32-second cycle, which translates to a calibration range of -487.1 ppm to +488.5 ppm with a resolution of about 0.954 ppm.

The formula to calculate the effective calibrated frequency ($F_{CAL}$) given the input frequency ($F_{RTCCLK}$) is as follows:

$$F_{CAL} = F_{RTCCLK} \times \left[1 + (CALT x 512 - CALM) / (2^{20} + CALM - CALP x 512)\right]$$
**Calibration when PREDIV_A<3**

The CALP bit cannot be set to 1 when the asynchronous prescaler value (PREDIV_A bits in RTC_PRER register) is less than 3. If CALP was already set to 1 and PREDIV_A bits are set to a value less than 3, CALP is ignored and the calibration operates as if CALP was equal to 0.

To perform a calibration with PREDIV_A less than 3, the synchronous prescaler value (PREDIV_S) should be reduced so that each second is accelerated by 8 RTCCLK clock cycles, which is equivalent to adding 256 clock cycles every 32 seconds. As a result, between 255 and 256 clock pulses (corresponding to a calibration range from 243.3 to 244.1 ppm) can effectively be added during each 32-second cycle using only the CALM bits.

With a nominal RTCCLK frequency of 32768 Hz, when PREDIV_A equals 1 (division factor of 2), PREDIV_S should be set to 16379 rather than 16383 (4 less). The only other interesting case is when PREDIV_A equals 0, PREDIV_S should be set to 32759 rather than 32767 (8 less).

If PREDIV_S is reduced in this way, the formula given the effective frequency of the calibrated input clock is as follows:

\[ F_{\text{CAL}} = F_{\text{RTCCLK}} \times \left[ 1 + \frac{(256 - \text{CALM})}{(2^{20} + \text{CALM} - 256)} \right] \]

In this case, CALM[7:0] equals 0x100 (the midpoint of the CALM range) is the correct setting if RTCCLK is exactly 32768.00 Hz.

**Verifying the RTC calibration**

RTC precision is ensured by measuring the precise frequency of RTCCLK and calculating the correct CALM value and CALP values. An optional 1 Hz output is provided to allow applications to measure and verify the RTC precision.

Measuring the precise frequency of the RTC over a limited interval can result in a measurement error of up to 2 RTCCLK clock cycles over the measurement period, depending on how the digital calibration cycle is aligned with the measurement period.

However, this measurement error can be eliminated if the measurement period is the same length as the calibration cycle period. In this case, the only error observed is the error due to the resolution of the digital calibration.

- By default, the calibration cycle period is 32 seconds.

Using this mode and measuring the accuracy of the 1 Hz output over exactly 32 seconds guarantees that the measure is within 0.477 ppm (0.5 RTCCLK cycles over 32 seconds, due to the limitation of the calibration resolution).

- CALW16 bit of the RTC_CALR register can be set to 1 to force a 16-second calibration cycle period.

In this case, the RTC precision can be measured during 16 seconds with a maximum error of 0.954 ppm (0.5 RTCCLK cycles over 16 seconds). However, since the calibration resolution is reduced, the long term RTC precision is also reduced to 0.954 ppm: CALM[0] bit is stuck at 0 when CALW16 is set to 1.

- CALW8 bit of the RTC_CALR register can be set to 1 to force a 8-second calibration cycle period.

In this case, the RTC precision can be measured during 8 seconds with a maximum error of 1.907 ppm (0.5 RTCCLK cycles over 8s). The long term RTC precision is also reduced to 1.907 ppm: CALM[1:0] bits are stuck at 00 when CALW8 is set to 1.
Re-calibration on-the-fly

The calibration register (RTC_CALR) can be updated on-the-fly while RTC_ISR/INITF=0, by using the follow process:

1. Poll the RTC_ISR/RECALPF (re-calibration pending flag).
2. If it is set to 0, write a new value to RTC_CALR, if necessary. RECALPF is then automatically set to 1.
3. Within three ck_apre cycles after the write operation to RTC_CALR, the new calibration settings take effect.

For code example refer to the Appendix section A.13.5: RTC calibration.

21.4.13 Time-stamp function

Time-stamp is enabled by setting the TSE bit of RTC_CR register to 1.

The calendar is saved in the time-stamp registers (RTC_TSSSR, RTC_TSTR, RTC_TSDR) when a time-stamp event is detected on the RTC_TS pin.

When a time-stamp event occurs, the time-stamp flag bit (TSF) in RTC_ISR register is set.

By setting the TSIE bit in the RTC_CR register, an interrupt is generated when a time-stamp event occurs.

If a new time-stamp event is detected while the time-stamp flag (TSF) is already set, the time-stamp overflow flag (TSOVF) flag is set and the time-stamp registers (RTC_TSTR and RTC_TSDR) maintain the results of the previous event.

Note: TSF is set 2 ck_apre cycles after the time-stamp event occurs due to synchronization process.

There is no delay in the setting of TSOVF. This means that if two time-stamp events are close together, TSOVF can be seen as ‘1’ while TSF is still ‘0’. As a consequence, it is recommended to poll TSOVF only after TSF has been set.

Caution: If a time-stamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set. To avoid masking a time-stamp event occurring at the same moment, the application must not write ‘0’ into TSF bit unless it has already read it to ‘1’.

Optionally, a tamper event can cause a time-stamp to be recorded. See the description of the TAMPTS control bit in Section 21.7.15: RTC tamper and alternate function configuration register (RTC_TAFCR).

21.4.14 Tamper detection

The RTC_TAMPx input events can be configured either for edge detection, or for level detection with filtering.

The tamper detection can be configured for the following purpose:

• generate an interrupt, capable to wake-up from Stop and Standby modes

Tamper detection initialization

Each input can be enabled by setting the corresponding TAMPxE bits to 1 in the RTC_TAFCR register.
Each RTC_TAMPx tamper detection input is associated with a flag TAMPxF in the RTC_ISR register. The TAMPxF flag is asserted after the tamper event on the pin, with the latency provided below:

- 3 ck_apre cycles when TAMPFLT differs from 0x0 (Level detection with filtering)
- 3 ck_apre cycles when TAMPTS=1 (Timestamp on tamper event)
- No latency when TAMPFLT=0x0 (Edge detection) and TAMPTS=0

A new tamper occurring on the same pin during this period and as long as TAMPxF is set cannot be detected.

By setting the TAMPIE bit in the RTC_TAFCR register, an interrupt is generated when a tamper detection event occurs.

**Timestamp on tamper event**

With TAMPTS set to ‘1’, any tamper event causes a timestamp to occur. In this case, either the TSF bit or the TSOVF bit are set in RTC_ISR, in the same manner as if a normal timestamp event occurs. The affected tamper flag register TAMPxF is set at the same time that TSF or TSOVF is set.

**Edge detection on tamper inputs**

If the TAMPFLT bits are "00", the RTC_TAMPx pins generate tamper detection events when either a rising edge or a falling edge is observed depending on the corresponding TAMPxTRG bit. The internal pull-up resistors on the RTC_TAMPx inputs are deactivated when edge detection is selected.

**Caution:** To avoid losing tamper detection events, the signal used for edge detection is logically ANDed with the corresponding TAMPxE bit in order to detect a tamper detection event in case it occurs before the RTC_TAMPx pin is enabled.

- When TAMPxTRG = 0: if the RTC_TAMPx is already high before tamper detection is enabled (TAMPxE bit set to 1), a tamper event is detected as soon as the RTC_TAMPx input is enabled, even if there was no rising edge on the RTC_TAMPx input after TAMPxE was set.
- When TAMPxTRG = 1: if the RTC_TAMPx is already low before tamper detection is enabled, a tamper event is detected as soon as the RTC_TAMPx input is enabled (even if there was no falling edge on the RTC_TAMPx input after TAMPxE was set.

**Level detection with filtering on RTC_TAMPx inputs**

Level detection with filtering is performed by setting TAMPFLT to a non-zero value. A tamper detection event is generated when either 2, 4, or 8 (depending on TAMPFLT) consecutive samples are observed at the level designated by the TAMPxTRG bits.

The RTC_TAMPx inputs are precharged through the I/O internal pull-up resistance before its state is sampled, unless disabled by setting TAMPPUDIS to 1. The duration of the precharge is determined by the TAMPPRCH bits, allowing for larger capacitances on the RTC_TAMPx inputs.

The trade-off between tamper detection latency and power consumption through the pull-up can be optimized by using TAMPFREQ to determine the frequency of the sampling for level detection.

**Note:** Refer to the datasheets for the electrical characteristics of the pull-up resistors.
For code example refer to the Appendix sections: \textit{A.13.6: RTC tamper and time stamp configuration} and \textit{A.13.7: RTC tamper and time stamp}.

\subsection*{21.4.15 Calibration clock output}

When the COE bit is set to 1 in the RTC\_CR register, a reference clock is provided on theRTC\_CALIB device output.

If the COSEL bit in the RTC\_CR register is reset and PREDIV\_A = 0x7F, the RTC\_CALIB frequency is \(f_{\text{RTCCLK}}/64\). This corresponds to a calibration output at 512 Hz for an RTCCCLK frequency at 32.768 kHz. The RTC\_CALIB duty cycle is irregular: there is a light jitter on falling edges. It is therefore recommended to use rising edges.

When COSEL is set and “PREDIV\_S+1” is a non-zero multiple of 256 (i.e: PREDIV\_S[7:0] = 0xFF), the RTC\_CALIB frequency is \(f_{\text{RTCCLK}}/(256 \times (PREDIV\_A+1))\). This corresponds to a calibration output at 1 Hz for prescaler default values (PREDIV\_A = Ox7F, PREDIV\_S = 0xFF), with an RTCCCLK frequency at 32.768 kHz. The 1 Hz output is affected when a shift operation is on going and may toggle during the shift operation (SHPF=1).

\textbf{Note:} \textit{When COSEL bit is cleared, the RTC\_CALIB output is the output of the 6th stage of the asynchronous prescaler.}

\textit{When COSEL bit is set, the RTC\_CALIB output is the output of the 8th stage of the synchronous prescaler.}

For code example refer to the Appendix section \textit{A.13.8: RTC clock output}.

\subsection*{21.4.16 Alarm output}

The OSEL[1:0] control bits in the RTC\_CR register are used to activate the alarm outputRTC\_ALARM, and to select the function which is output. These functions reflect the contents of the corresponding flags in the RTC\_ISR register.

The polarity of the output is determined by the POL control bit in RTC\_CR so that the opposite of the selected flag bit is output when POL is set to 1.

\textbf{Alarm output}

The RTC\_ALARM pin can be configured in output open drain or output push-pull usingRTC\_TAFCR register.

\textbf{Note:} \textit{Once the RTC\_ALARM output is enabled, it has priority over RTC\_CALIB (COE bit is don't care and must be kept cleared).}
21.5 RTC low-power modes

Table 65. Effect of low-power modes on RTC

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>No effect; RTC interrupts cause the device to exit the Sleep mode.</td>
</tr>
<tr>
<td>Stop</td>
<td>The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC wake-up cause the device to exit the Stop mode.</td>
</tr>
<tr>
<td>Standby</td>
<td>The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC wake-up cause the device to exit the Standby mode.</td>
</tr>
</tbody>
</table>

21.6 RTC interrupts

All RTC interrupts are connected to the EXTI controller. Refer to .

To enable RTC interrupt(s), the following sequence is required:

1. Configure and enable the NVIC line(s) corresponding to the RTC event(s) in interrupt mode and select the rising edge sensitivity.
2. Configure and enable the RTC IRQ channel in the NVIC.
3. Configure the RTC to generate RTC interrupt(s).

Table 66. Interrupt control bits

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
<th>Exit the Sleep mode</th>
<th>Exit the Stop mode</th>
<th>Exit the Standby mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm A</td>
<td>ALRAF</td>
<td>ALRAIE</td>
<td>Yes</td>
<td>Yes(1)</td>
<td>Yes(1)</td>
</tr>
<tr>
<td>RTC_TS input (timestamp)</td>
<td>TSF</td>
<td>TSIE</td>
<td>Yes</td>
<td>Yes(1)</td>
<td>Yes(1)</td>
</tr>
<tr>
<td>RTC_TAMP1 input detection</td>
<td>TAMP1F</td>
<td>TAMPIE</td>
<td>Yes</td>
<td>Yes(1)</td>
<td>Yes(1)</td>
</tr>
<tr>
<td>RTC_TAMP2 input detection</td>
<td>TAMP2F</td>
<td>TAMPIE</td>
<td>Yes</td>
<td>Yes(1)</td>
<td>Yes(1)</td>
</tr>
</tbody>
</table>

1. Wake-up from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

21.7 RTC registers

Refer to  Section 1.2 on page 33 of the reference manual for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by words (32-bit).

21.7.1 RTC time register (RTC_TR)

The RTC_TR is the calendar time shadow register. This register must be written in initialization mode only. Refer to  Calendar initialization and configuration on page 492 and  Reading the calendar on page 493.
This register is write protected. The write access procedure is described in *RTC register write protection on page 492*.

Address offset: 0x00

RTC domain reset value: 0x0000 0000

System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

<table>
<thead>
<tr>
<th>Address offset</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x0000 0000</td>
<td>System reset value</td>
</tr>
</tbody>
</table>

### 21.7.2 RTC date register (RTC_DR)

The RTC_DR is the calendar date shadow register. This register must be written in initialization mode only. Refer to *Calendar initialization and configuration on page 492* and *Reading the calendar on page 493*.

This register is write protected. The write access procedure is described in *RTC register write protection on page 492*.

Address offset: 0x04

RTC domain reset value: 0x0000 2101

System reset: 0x0000 2101 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

<table>
<thead>
<tr>
<th>Address offset</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04</td>
<td>0x0000 2101</td>
<td>System reset value</td>
</tr>
</tbody>
</table>

### Bit Descriptions

- **PM**: AM/PM notation
  
  0: AM or 24-hour format
  1: PM

- **HT[1:0]**: Hour tens in BCD format

- **HU[3:0]**: Hour units in BCD format

- **MNT[2:0]**: Minute tens in BCD format

- **MNU[3:0]**: Minute units in BCD format

- **ST[2:0]**: Second tens in BCD format

- **SU[3:0]**: Second units in BCD format
Bits 31:24  Reserved, must be kept at reset value.

Bits 23:20  **YT[3:0]**: Year tens in BCD format

Bits 19:16  **YU[3:0]**: Year units in BCD format

Bits 15:13  **WDU[2:0]**: Week day units

000: forbidden
001: Monday
...
111: Sunday

Bit 12  **MT**: Month tens in BCD format

Bits 11:8  **MU[3:0]**: Month units in BCD format

Bits 7:6  Reserved, must be kept at reset value.

Bits 5:4  **DT[1:0]**: Date tens in BCD format

Bits 3:0  **DU[3:0]**: Date units in BCD format
### 21.7.3 RTC control register (RTC_CR)

Address offset: 0x08

RTC domain reset value: 0x0000 0000

System reset: not affected

<table>
<thead>
<tr>
<th>Bit 31:24</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
</table>

**Bit 23** **COE**: Calibration output enable

This bit enables the RTC_CALIB output

0: Calibration output disabled

1: Calibration output enabled

**Bits 22:21** **OSEL[1:0]**: Output selection

These bits are used to select the flag to be routed to RTC_ALARM output

- 00: Output disabled
- 01: Alarm A output enabled
- 10: Reserved
- 11: Wake-up output enabled

**Bit 20** **POL**: Output polarity

This bit is used to configure the polarity of RTC_ALARM output

0: The pin is high when ALRAF/WUTF is asserted (depending on OSEL[1:0])

1: The pin is low when ALRAF/WUTF is asserted (depending on OSEL[1:0]).

**Bit 19** **COSEL**: Calibration output selection

When COE=1, this bit selects which signal is output on RTC_CALIB.

- 0: Calibration output is 512 Hz (with default prescaler setting)
- 1: Calibration output is 1 Hz (with default prescaler setting)

These frequencies are valid for RTCCLK at 32.768 kHz and prescalers at their default values (PREDIV_A=127 and PREDIV_S=255). Refer to Section 21.4.15: Calibration clock output

**Bit 18** **BKP**: Backup

This bit can be written by the user to memorize whether the daylight saving time change has been performed or not.

**Bit 17** **SUB1H**: Subtract 1 hour (winter time change)

When this bit is set, 1 hour is subtracted to the calendar time if the current hour is not 0. This bit is always read as 0.

Setting this bit has no effect when current hour is 0.

0: No effect

1: Subtracts 1 hour to the current time. This can be used for winter time change outside initialization mode.
Bit 16  **ADD1H**: Add 1 hour (summer time change)
When this bit is set, 1 hour is added to the calendar time. This bit is always read as 0.
  0: No effect
  1: Adds 1 hour to the current time. This can be used for summer time change outside initialization mode.

Bit 15  **TSIE**: Time-stamp interrupt enable
  0: Time-stamp Interrupt disable
  1: Time-stamp Interrupt enable

Bit 14  **WUTIE**: Wake-up timer interrupt enable
  0: Wake-up timer interrupt disabled
  1: Wake-up timer interrupt enabled

Bit 13  Reserved, must be kept at reset value.

Bit 12  **ALRAIE**: Alarm A interrupt enable
  0: Alarm A interrupt disabled
  1: Alarm A interrupt enabled

Bit 11  **TSE**: timestamp enable
  0: timestamp disable
  1: timestamp enable

Bit 10  **WUTE**: Wake-up timer enable
  0: Wake-up timer disabled
  1: Wake-up timer enabled

  *Note: When the wake-up timer is disabled, wait for WUTWF=1 before enabling it again.*

Bit 9  Reserved, must be kept at reset value.

Bit 8  **ALRAE**: Alarm A enable
  0: Alarm A disabled
  1: Alarm A enabled

Bit 7  Reserved, must be kept at reset value.

Bit 6  **FMT**: Hour format
  0: 24 hour/day format
  1: AM/PM hour format

Bit 5  **BYPISHAD**: Bypass the shadow registers
  0: Calendar values (when reading from RTC_SSR, RTC_TR, and RTC_DR) are taken from the shadow registers, which are updated once every two RTCCCLK cycles.
  1: Calendar values (when reading from RTC_SSR, RTC_TR, and RTC_DR) are taken directly from the calendar counters.

  *Note: If the frequency of the APB clock is less than seven times the frequency of RTCCLK, BYPISHAD must be set to ‘1’.*
Bit 4 **REFCKON**: RTC_REFIN reference clock detection enable (50 or 60 Hz)
- 0: RTC_REFIN detection disabled
- 1: RTC_REFIN detection enabled
*Note: PREDIV_S must be 0x00FF.*

Bit 3 **TSEDGE**: Time-stamp event active edge
- 0: RTC_TS input rising edge generates a time-stamp event
- 1: RTC_TS input falling edge generates a time-stamp event
TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting.

Bits 2:0 **WUCKSEL[2:0]**: Wake-up clock selection
- 000: RTC/16 clock is selected
- 001: RTC/8 clock is selected
- 010: RTC/4 clock is selected
- 011: RTC/2 clock is selected
- 10x: ck_spre (usually 1 Hz) clock is selected
- 11x: ck_spre (usually 1 Hz) clock is selected and 2^16 is added to the WUT counter value (see note below)

*Note: Bits 7, 6 and 4 of this register can be written in initialization mode only (RTC_ISR/INITF = 1).*

**WUT = Wake-up unit counter value. WUT = (0x0000 to 0xFFFF) + 0x10000 added when WUCKSEL[2:1 = 1].**

**Bits 2 to 0 of this register can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1.**

It is recommended not to change the hour during the calendar hour increment as it could mask the incrementation of the calendar hour.

ADD1H and SUB1H changes are effective in the next second.

This register is write protected. The write access procedure is described in RTC register write protection on page 492.

**Caution:** TSE must be reset when TSEDGE is changed to avoid spuriously setting of TSF.
### 21.7.4 RTC initialization and status register (RTC_ISR)

This register is write protected (except for RTC_ISR[13:8] bits). The write access procedure is described in *RTC register write protection on page 492*.

**Address offset:** 0x0C

**RTC domain reset value:** 0x0000 0007

**System reset:** not affected except INIT, INITF, and RSF bits which are cleared to ‘0’

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>TAMP2F</td>
<td>TAMP1F</td>
<td>TSOVF</td>
<td>TSF</td>
<td>WUTF</td>
<td>ALRAF</td>
<td>INIT</td>
<td>INITF</td>
<td>RSF</td>
<td>INITS</td>
<td>SHPF</td>
<td>WUTWF</td>
<td>ALRAWF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rc_w0</td>
<td>rw</td>
<td>r</td>
<td>rc_w0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:17 Reserved, must be kept at reset value.

**Bit 16 RECALPF:** Recalibration pending Flag

The RECALPF status flag is automatically set to ‘1’ when software writes to the RTC_CALR register, indicating that the RTC_CALR register is blocked. When the new calibration settings are taken into account, this bit returns to ‘0’. Refer to *Re-calibration on-the-fly*.

**Bit 15** Reserved, must be kept at reset value.

**Bit 14 TAMP2F:** RTC_TAMP2 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP2 input.

It is cleared by software writing 0

**Bit 13 TAMP1F:** RTC_TAMP1 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP1 input.

It is cleared by software writing 0

**Bit 12 TSOVF:** Time-stamp overflow flag

This flag is set by hardware when a time-stamp event occurs while TSF is already set.

This flag is cleared by software by writing 0. It is recommended to check and then clear TSOVF only after clearing the TSF bit. Otherwise, an overflow might not be noticed if a time-stamp event occurs immediately before the TSF bit is cleared.

**Bit 11 TSF:** Time-stamp flag

This flag is set by hardware when a time-stamp event occurs.

This flag is cleared by software by writing 0.

**Bit 10 WUTF:** Wake-up timer flag

This flag is set by hardware when the wake-up auto-reload counter reaches 0.

This flag is cleared by software by writing 0.

This flag must be cleared by software at least 1.5 RTCCLK periods before WUTF is set to 1 again.

**Bit 9** Reserved, must be kept at reset value.
Bit 8 **ALRAF**: Alarm A flag
This flag is set by hardware when the time/date registers (RTC_TR and RTC_DR) match the Alarm A register (RTC_ALRMAR).
This flag is cleared by software by writing 0.

Bit 7 **INIT**: Initialization mode
0: Free running mode
1: Initialization mode used to program time and date register (RTC_TR and RTC_DR), and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Bit 6 **INITF**: Initialization flag
When this bit is set to 1, the RTC is in initialization state, and the time, date and prescaler registers can be updated.
0: Calendar registers update is not allowed
1: Calendar registers update is allowed

Bit 5 **RSF**: Registers synchronization flag
This bit is set by hardware each time the calendar registers are copied into the shadow registers (RTC_SSR, RTC_TR and RTC_DR). This bit is cleared by hardware in initialization mode, while a shift operation is pending (SHPF=1), or when in bypass shadow register mode (BYPSHAD=1). This bit can also be cleared by software.
It is cleared either by software or by hardware in initialization mode.
0: Calendar shadow registers not yet synchronized
1: Calendar shadow registers synchronized

Bit 4 **INITS**: Initialization status flag
This bit is set by hardware when the calendar year field is different from 0 (RTC domain reset state).
0: Calendar has not been initialized
1: Calendar has been initialized

Bit 3 **SHPF**: Shift operation pending
0: No shift operation is pending
1: A shift operation is pending
This flag is set by hardware as soon as a shift operation is initiated by a write to the RTC_SHIFTR register. It is cleared by hardware when the corresponding shift operation has been executed. Writing to the SHPF bit has no effect.

Bit 2 **WUTWF**: Wake-up timer write flag
This bit is set by hardware up to 2 RTCCLK cycles after the WUTE bit has been set to 0 in RTC_CR, and is cleared up to 2 RTCCLK cycles after the WUTE bit has been set to 1. The wake-up timer values can be changed when WUTE bit is cleared and WUTWF is set.
0: Wake-up timer configuration update not allowed
1: Wake-up timer configuration update allowed

Bit 1 Reserved, must be kept at reset value.

Bit 0 **ALRAWF**: Alarm A write flag
This bit is set by hardware when Alarm A values can be changed, after the ALRAE bit has been set to 0 in RTC_CR.
It is cleared by hardware in initialization mode.
0: Alarm A update not allowed
1: Alarm A update allowed

**Note:** The bits ALRAF, WUTF and TSF are cleared 2 APB clock cycles after programming them to 0.
21.7.5 **RTC prescaler register (RTC_PRER)**

This register must be written in initialization mode only. The initialization must be performed in two separate write accesses. Refer to *Calendar initialization and configuration on page 492*.

This register is write protected. The write access procedure is described in *RTC register write protection on page 492*.

Address offset: 0x10

RTC domain reset value: 0x007F 00FF

System reset: not affected

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:23  Reserved, must be kept at reset value.

Bits 22:16  **PREDIV_A[6:0]**: Asynchronous prescaler factor

This is the asynchronous division factor:
\[
ck_{apre} \text{ frequency} = \frac{\text{RTCCLK frequency}}{PREDIV_A+1}
\]

Bit 15  Reserved, must be kept at reset value.

Bits 14:0  **PREDIV_S[14:0]**: Synchronous prescaler factor

This is the synchronous division factor:
\[
ck_{spre} \text{ frequency} = \frac{ck_{apre} \text{ frequency}}{PREDIV_S+1}
\]
21.7.6 RTC wake-up timer register (RTC_WUTR)

This register can be written only when WUTWF is set to 1 in RTC_ISR.

This register is write protected. The write access procedure is described in RTC register write protection on page 492.

Address offset: 0x14

RTC domain reset value: 0x0000 FFFF

System reset: not affected

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

WUT[15:0]

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **WUT[15:0]:** Wake-up auto-reload value bits

When the wake-up timer is enabled (WUTE set to 1), the WUTF flag is set every (WUT[15:0] + 1) \( ck\_wut \) cycles. The \( ck\_wut \) period is selected through WUCKSEL[2:0] bits of the RTC_CR register.

When WUCKSEL[2] = 1, the wake-up timer becomes 17-bits and WUCKSEL[1] effectively becomes WUT[16] the most-significant bit to be reloaded into the timer.

The first assertion of WUTF occurs (WUT+1) \( ck\_wut \) cycles after WUTE is set. Setting WUT[15:0] to 0x0000 with WUCKSEL[2:0] =011 (RTCCLK/2) is forbidden.
21.7.7 RTC alarm A register (RTC_ALRMAR)

This register can be written only when ALRAWF is set to 1 in RTC_ISR, or in initialization mode.

This register is write protected. The write access procedure is described in RTC register write protection on page 492.

Address offset: 0x1C
RTC domain reset value: 0x0000 0000
System reset: not affected

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>MSK4</td>
<td>Alarm A date mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Alarm A set if the date/day match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Date/day don’t care in Alarm A comparison</td>
</tr>
<tr>
<td>30</td>
<td>WDSEL</td>
<td>Week day selection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: DU[3:0] represents the date units</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: DU[3:0] represents the week day. DT[1:0] is don’t care.</td>
</tr>
<tr>
<td>29-28</td>
<td>DT[1:0]</td>
<td>Date tens in BCD format.</td>
</tr>
<tr>
<td>27-24</td>
<td>DU[3:0]</td>
<td>Date units or day in BCD format.</td>
</tr>
<tr>
<td>23</td>
<td>MSK3</td>
<td>Alarm A hours mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Alarm A set if the hours match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Hours don’t care in Alarm A comparison</td>
</tr>
<tr>
<td>22</td>
<td>PM</td>
<td>AM/PM notation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: AM or 24-hour format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: PM</td>
</tr>
<tr>
<td>21-20</td>
<td>HT[1:0]</td>
<td>Hour tens in BCD format.</td>
</tr>
<tr>
<td>19-16</td>
<td>HU[3:0]</td>
<td>Hour units in BCD format.</td>
</tr>
<tr>
<td>15</td>
<td>MSK2</td>
<td>Alarm A minutes mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Alarm A set if the minutes match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Minutes don’t care in Alarm A comparison</td>
</tr>
<tr>
<td>14-12</td>
<td>MNT[2:0]</td>
<td>Minute tens in BCD format.</td>
</tr>
<tr>
<td>11-8</td>
<td>MNU[3:0]</td>
<td>Minute units in BCD format.</td>
</tr>
<tr>
<td>7</td>
<td>MSK1</td>
<td>Alarm A seconds mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Alarm A set if the seconds match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Seconds don’t care in Alarm A comparison</td>
</tr>
<tr>
<td>6-4</td>
<td>ST[2:0]</td>
<td>Second tens in BCD format.</td>
</tr>
<tr>
<td>3-0</td>
<td>SU[3:0]</td>
<td>Second units in BCD format.</td>
</tr>
</tbody>
</table>
21.7.8 RTC write protection register (RTC_WPR)

Address offset: 0x24
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 KEY[7:0]: Write protection key
This byte is written by software.
Reading this byte always returns 0x00.
Refer to RTC register write protection for a description of how to unlock RTC register write protection.

21.7.9 RTC sub second register (RTC_SSR)

Address offset: 0x28
RTC domain reset value: 0x0000 0000
System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

SS[15:0]

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 SS[15:0]: Sub second value
SS[15:0] is the value in the synchronous prescaler counter. The fraction of a second is given by the formula below:
Second fraction = (PREDIV_S - SS) / (PREDIV_S + 1)
Note: SS can be larger than PREDIV_S only after a shift operation. In that case, the correct time/date is one second less than as indicated by RTC_TR/RTC_DR.
21.7.10 RTC shift control register (RTC_SHIFTR)

This register is write protected. The write access procedure is described in RTC register write protection on page 492.

Address offset: 0x2C

RTC domain reset value: 0x0000 0000

System reset: not affected

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBFS[14:0]</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>

Bit 31 ADD1S: Add one second
0: No effect
1: Add one second to the clock/calendar

This bit is write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC_ISR).

This function is intended to be used with SUBFS (see description below) in order to effectively add a fraction of a second to the clock in an atomic operation.

Bits 30:15 Reserved, must be kept at reset value.

Bits 14:0 SUBFS[14:0]: Subtract a fraction of a second

These bits are write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC_ISR).

The value which is written to SUBFS is added to the synchronous prescaler counter. Since this counter counts down, this operation effectively subtracts from (delays) the clock by:

Delay (seconds) = SUBFS / (PREDIV_S + 1)

A fraction of a second can effectively be added to the clock (advancing the clock) when the ADD1S function is used in conjunction with SUBFS, effectively advancing the clock by:

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1))).

Note: Writing to SUBFS causes RSF to be cleared. Software can then wait until RSF=1 to be sure that the shadow registers have been updated with the shifted time.
21.7.11 **RTC timestamp time register (RTC_TSTR)**

The content of this register is valid only when TSF is set to 1 in RTC_ISR. It is cleared when TSF bit is reset.

- **Address offset:** 0x30
- **RTC domain reset value:** 0x0000 0000
- **System reset:** not affected

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:23</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td><strong>PM:</strong> AM/PM notation</td>
<td>0: AM or 24-hour format 1: PM</td>
</tr>
<tr>
<td>21:16</td>
<td><strong>HT[1:0]:</strong> Hour tens in BCD format.</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
<tr>
<td>14:12</td>
<td><strong>MNT[2:0]:</strong> Minute tens in BCD format.</td>
<td></td>
</tr>
<tr>
<td>11:8</td>
<td><strong>MNU[3:0]:</strong> Minute units in BCD format.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Reserved, must be kept at reset value.</td>
<td></td>
</tr>
<tr>
<td>6:4</td>
<td><strong>ST[2:0]:</strong> Second tens in BCD format.</td>
<td></td>
</tr>
<tr>
<td>3:0</td>
<td><strong>SU[3:0]:</strong> Second units in BCD format.</td>
<td></td>
</tr>
</tbody>
</table>
21.7.12 RTC timestamp date register (RTC_TSDR)

The content of this register is valid only when TSF is set to 1 in RTC_ISR. It is cleared when TSF bit is reset.

Address offset: 0x34

RTC domain reset value: 0x0000 0000

System reset: not affected

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:13 **WDU[2:0]**: Week day units

Bit 12 **MT**: Month tens in BCD format

Bits 11:8 **MU[3:0]**: Month units in BCD format

Bits 7:6 Reserved, must be kept at reset value.

Bits 5:4 **DT[1:0]**: Date tens in BCD format

Bits 3:0 **DU[3:0]**: Date units in BCD format
21.7.13 RTC time-stamp sub second register (RTC_TSSSR)

The content of this register is valid only when RTC_ISR/TSF is set. It is cleared when the RTC_ISR/TSF bit is reset.

Address offset: 0x38

RTC domain reset value: 0x0000 0000

System reset: not affected

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SS[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>r r r r r r r r r r r r r r r r</td>
</tr>
</tbody>
</table>

Bits 31:16  Reserved, must be kept at reset value.

Bits 15:0  SS[15:0]: Sub second value

SS[15:0] is the value of the synchronous prescaler counter when the timestamp event occurred.
### 21.7.14 RTC calibration register (RTC_CALR)

This register is write protected. The write access procedure is described in *RTC register write protection on page 492.*

Address offset: 0x3C

RTC domain reset value: 0x0000 0000

System reset: not affected

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:16 Reserved, must be kept at reset value.

- **Bit 15 CalP:** Increase frequency of RTC by 488.5 ppm
  0: No RTCCLK pulses are added.
  1: One RTCCLK pulse is effectively inserted every \(2^{11}\) pulses (frequency increased by 488.5 ppm).

This feature is intended to be used in conjunction with CALM, which lowers the frequency of the calendar with a fine resolution. If the input frequency is 32768 Hz, the number of RTCCLK pulses added during a 32-second window is calculated as follows: \((512 \times \text{CalP}) - \text{CALM}\).

Refer to *Section 21.4.12: RTC smooth digital calibration.*

- **Bit 14 CalW8:** Use an 8-second calibration cycle period
  When CalW8 is set to ‘1’, the 8-second calibration cycle period is selected.
  *Note:* CalM[1:0] are stuck at “00” when CalW8=’1’. Refer to *Section 21.4.12: RTC smooth digital calibration.*

- **Bit 13 CalW16:** Use a 16-second calibration cycle period
  When CalW16 is set to ‘1’, the 16-second calibration cycle period is selected. This bit must not be set to ‘1’ if CalW8=1.
  *Note:* CalM[0] is stuck at ‘0’ when CalW16=’1’. Refer to *Section 21.4.12: RTC smooth digital calibration.*

Bits 12:9 Reserved, must be kept at reset value.

- **Bits 8:0 CalM[8:0]:** Calibration minus
  The frequency of the calendar is reduced by masking CALM out of \(2^{20}\) RTCCLK pulses (32 seconds if the input frequency is 32768 Hz). This decreases the frequency of the calendar with a resolution of 0.9537 ppm.

To increase the frequency of the calendar, this feature should be used in conjunction with CalP. See *Section 21.4.12: RTC smooth digital calibration on page 496.*
21.7.15  RTC tamper and alternate function configuration register
(RTC_TAFCR)

Address offset: 0x40
RTC domain reset value: 0x0000 0000
System reset: not affected

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TAMPPUDIS</td>
<td>TAMPPRCH [1:0]</td>
<td>TAMPFREQ[2:0]</td>
<td>TAMPTY [1:0]</td>
<td>TAMP2 TRG</td>
<td>TAMP2 E</td>
<td>TAMP1 IE</td>
<td>TAMP1 TRG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:24  Reserved, must be kept at reset value.

Bit 23  PC15MODE: PC15 mode
0: PC15 is controlled by the GPIO configuration registers. Consequently PC15 is floating in Standby mode.
1: PC15 is forced to push-pull output if LSE is disabled.

Bit 22  PC15VALUE: PC15 value
If the LSE is disabled and PC15MODE = 1, PC15VALUE configures the PC15 output data.

Bit 21  PC14MODE: PC14 mode
0: PC14 is controlled by the GPIO configuration registers. Consequently PC14 is floating in Standby mode.
1: PC14 is forced to push-pull output if LSE is disabled.

Bit 20  PC14VALUE: PC14 value
If the LSE is disabled and PC14MODE = 1, PC14VALUE configures the PC14 output data.

Bit 19  PC13MODE: PC13 mode
0: PC13 is controlled by the GPIO configuration registers. Consequently PC13 is floating in Standby mode.
1: PC13 is forced to push-pull output if all RTC functions are disabled.

Bit 18  PC13VALUE: RTC_ALARM output type/PC13 value
If PC13 is used to output RTC_ALARM, PC13VALUE configures the output configuration:
0: RTC_ALARM is an open-drain output
1: RTC_ALARM is a push-pull output
If all RTC functions are disabled and PC13MODE = 1, PC13VALUE configures the PC13 output data.

Bits 17:16  Reserved, must be kept at reset value.

Bit 15  TAMPPUDIS: RTC_TAMPx pull-up disable
This bit determines if each of the RTC_TAMPx pins are pre-charged before each sample.
0: Precharge RTC_TAMPx pins before sampling (enable internal pull-up)
1: Disable precharge of RTC_TAMPx pins.
Bits 14:13 **TAMPPRCH[1:0]:** RTC_TAMPx precharge duration
These bits determine the duration of time during which the pull-up is activated before each sample. TAMPPRCH is valid for each of the RTC_TAMPx inputs.
- 0x0: 1 RTCCLK cycle
- 0x1: 2 RTCCLK cycles
- 0x2: 4 RTCCLK cycles
- 0x3: 8 RTCCLK cycles

Bits 12:11 **TAMPFLT[1:0]:** RTC_TAMPx filter count
These bits determine the number of consecutive samples at the specified level (TAMP_TRG) needed to activate a Tamper event. TAMPFLT is valid for each of the RTC_TAMPx inputs.
- 0x0: Tamper event is activated on edge of RTC_TAMPx input transitions to the active level (no internal pull-up on RTC_TAMPx input).
- 0x1: Tamper event is activated after 2 consecutive samples at the active level.
- 0x2: Tamper event is activated after 4 consecutive samples at the active level.
- 0x3: Tamper event is activated after 8 consecutive samples at the active level.

Bits 10:8 **TAMPFREQ[2:0]:** Tamper sampling frequency
Determines the frequency at which each of the RTC_TAMPx inputs are sampled.
- 0x0: RTCCLK / 32768 (1 Hz when RTCCLK = 32768 Hz)
- 0x1: RTCCLK / 16384 (2 Hz when RTCCLK = 32768 Hz)
- 0x2: RTCCLK / 8192 (4 Hz when RTCCLK = 32768 Hz)
- 0x3: RTCCLK / 4096 (8 Hz when RTCCLK = 32768 Hz)
- 0x4: RTCCLK / 2048 (16 Hz when RTCCLK = 32768 Hz)
- 0x5: RTCCLK / 1024 (32 Hz when RTCCLK = 32768 Hz)
- 0x6: RTCCLK / 512 (64 Hz when RTCCLK = 32768 Hz)
- 0x7: RTCCLK / 256 (128 Hz when RTCCLK = 32768 Hz)

Bit 7 **TAMPTS:** Activate timestamp on tamper detection event
- 0: Tamper detection event does not cause a timestamp to be saved
- 1: Save timestamp on tamper detection event
TAMPTS is valid even if TSE=0 in the RTC_CR register.

Bits 6:5 Reserved, must be kept at reset value.

Bit 4 **TAMP2TRG:** Active level for RTC_TAMP2 input
- If TAMPFLT != 00:
  - 0: RTC_TAMP2 input staying low triggers a tamper detection event.
  - 1: RTC_TAMP2 input staying high triggers a tamper detection event.
- If TAMPFLT = 00:
  - 0: RTC_TAMP2 input rising edge triggers a tamper detection event.
  - 1: RTC_TAMP2 input falling edge triggers a tamper detection event.

Bit 3 **TAMP2E:** RTC_TAMP2 input detection enable
- 0: RTC_TAMP2 detection disabled
- 1: RTC_TAMP2 detection enabled
Bit 2 **TAMPIE**: Tamper interrupt enable
- 0: Tamper interrupt disabled
- 1: Tamper interrupt enabled.

Bit 1 **TAMP1TRG**: Active level for RTC_TAMP1 input
- If TAMPFLT != 00
  - 0: RTC_TAMP1 input staying low triggers a tamper detection event.
  - 1: RTC_TAMP1 input staying high triggers a tamper detection event.
- If TAMPFLT = 00:
  - 0: RTC_TAMP1 input rising edge triggers a tamper detection event.
  - 1: RTC_TAMP1 input falling edge triggers a tamper detection event.

Bit 0 **TAMP1E**: RTC_TAMP1 input detection enable
- 0: RTC_TAMP1 detection disabled
- 1: RTC_TAMP1 detection enabled

**Caution:** When TAMPFLT = 0, TAMPxE must be reset when TAMPxTRG is changed to avoid spuriously setting TAMPxF.
21.7.16 RTC alarm A sub second register (RTC_ALRMASSR)

This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

This register is write protected. The write access procedure is described in RTC register write protection on page 492

Address offset: 0x44
RTC domain reset value: 0x0000 0000
System reset: not affected

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 MASKSS[3:0]: Mask the most-significant bits starting at this bit
- 0: No comparison on sub seconds for Alarm A. The alarm is set when the seconds unit is incremented (assuming that the rest of the fields match).
- 1: SS[14:1] are don’t care in Alarm A comparison. Only SS[0] is compared.
- ...
- 15: All 15 SS bits are compared and must match to activate alarm.

The overflow bits of the synchronous counter (bits 15) is never compared. This bit can be different from 0 only after a shift operation.

Bits 23:15 Reserved, must be kept at reset value.

Bits 14:0 SS[14:0]: Sub seconds value
This value is compared with the contents of the synchronous prescaler counter to determine if Alarm A is to be activated. Only bits 0 up MASKSS-1 are compared.

21.7.17 RTC register map

Table 67. RTC register map and reset values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

Reset value

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RTC_TR</td>
</tr>
</tbody>
</table>

Reset value
0x38

522/775

RTC_TSSSR
0
0
0
0
0
0

Res.

Res.

Reset value

RM0360 Rev 5

WDU[1:0]

MT

0
0
0
0
0
0
0
0

RTC_WPR
Res.
Res.
Res.
Res.
Res.
Res.
Res.
Res.
Res.

HU[3:0]
MSK2

1
1

Res.
Res.

Reset value

Reset value

HU[3:0]

Reset value
0

1

0

0

0

0

1

0

0

0

0

0

0
0
0
0

PREDIV_A[6:0]

0

1

0

0

0

0

0

1

0

0

0

0

0

MNT[2:0]

0

0

0

0

0

0

1
1

0

0

0

0

0
0

0

0

0
0

1
1

MNU[3:0]

Reset value

0
0

0
0

MNU[3:0]

MU[3:0]

0

1

1

0

0

0

0

0

0

0

PREDIV_S[14:0]
1

WUT[15:0]
1

0

0

0

0

0

0

0
0
0
0
0

ST[2:0]
0

0

0

0

0

ST[2:0]

0

WUCKSEL[2:0]

TSEDGE

0
0

0
0
0
1

1
1
1
1
1
1

1
1
1
1
1
1

0

0
0

Res.

REFCKON

0

ALRAWF

SHPF

0

WUT WF

0

INITS

BYPSHAD

0

FMT

Res.

1

RSF

INIT
INITF

0

Res.

0

ALRAF

0

ALRAE

WUTE

0

Res.

0
WUTF

.TAMP2F

TSE

Res.
0

0
TSF

0

1

Res.

0

ALRAIE

0
TSOVF

TSIE
WUTIE

0
TAMP1F

ADD1H

0
RECALPF

POL
COSEL
BKP

0

SUB1H

0

Res.

0

Res.

0

MSK1

1

Res.

Res.

Res.

0

DT[1:0]

0

Res.

1

Res.

OSEL[1:0]

0

Res.

0

Res.

1

Res.

Res.

0

Res.

Res.

MT

DT[1:0]

Res.

Res.

MU[3:0]

Res.

0

Res.

Res.

COE

Res.

Res.

Res.

Res.

Res.

WDU[2:0]

Res.

0

Res.

1

Res.

0

MNT[2:0]

Res.

0

HT[1:0]

1

Res.

Res.

Res.

Res.

Res.

Res.

Res.

Res.

Res.

Res.

YU[3:0]

Res.

Res.

Res.

Res.

0

Res.

Res.

Reset value

Res.

Res.

Res.

0

Res.

Res.

Res.

Res.

0

Res.

0

Res.

0

Res.

Res.

Res.

Res.

0

Res.

Res.

0

Res.

Res.

Res.

Res.

0

Res.

Res.

PM

0

Res.

Res.

Res.

Res.

Res.

0

Res.

0

Res.

Res.

Res.

0

Res.

Res.

Res.

Res.

Res.

Res.

YT[3:0]

Res.

0

Res.

Res.

Res.

0

Res.

Res.

Res.

Res.

Res.

0

Res.

HT[1:0]

PM

0

Res.

Res.

Res.

0

Res.

0

Res.

Reset value

Res.

MSK3

0

Res.

Res.

Res.

Reset value

Res.

Res.

Res.

Res.

Res.

0
DT[1:0]

0

Res.

0

Res.

Res.

Res.

Res.

Res.

Res.

Res.

Res.

MSK4
WDSEL

Reset value
DU[3:0]

Res.

Res.

Res.

Res.

Res.

Res.

Res.

RTC_ALRMAR

Res.

Reset value

Res.

Res.

Res.

Res.

Res.

Res.

Res.

Res.

Reset value

Res.

Res.

Res.

Res.

RTC_TSDR

Res.

RTC_TSTR

Res.

0

Res.

0x34
Reset value

Res.

0x30
RTC_SHIFTR

Res.

0x2C
RTC_SSR

Res.

0x28
RTC_WUTR

Res.

0x24
RTC_PRER

Res.

0x1C

ADD1S

0x14
RTC_ISR

Res.

0x10

Res.

0x0C
RTC_CR

Res.

0x08
RTC_DR

Res.

0x04

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

Register
name

Res.

Offset

Res.

Real-time clock (RTC)
RM0360

Table 67. RTC register map and reset values (continued)

DU[3:0]
0

0

1

1

SU[3:0]
0
0
0

KEY

SS[15:0]

SUBFS[14:0]
0
0
0
0
0
0
0

0
0
0
0
0
0
0

0
0
0
0
0
0

SU[3:0]

0

DU[3:0]

SS[15:0]

0

0

0

0

0

0

0

0

0

0

0

0


Table 67. RTC register map and reset values (continued)

| Offset | Register name | 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  |
|--------|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0x3C   | RTC_CALR      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x40   | RTC_TAFCR     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | 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  | 0  |
| 0x44   | RTC_ALRMASSR  | MASKSS [3:0] |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x4C   | RTC_OR        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | Reset value   | 0  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Refer to Section 2.2 on page 37 for the register boundary addresses.
22 Inter-integrated circuit (I2C) interface

22.1 Introduction

The I\textsuperscript{2}C (inter-integrated circuit) bus interface handles communications between the microcontroller and the serial I\textsuperscript{2}C bus. It provides multimaster capability, and controls all I\textsuperscript{2}C bus-specific sequencing, protocol, arbitration and timing. It supports Standard-mode (Sm), Fast-mode (Fm) and Fast-mode Plus (Fm+).

The I\textsuperscript{2}C bus interface is also SMBus (system management bus) and PMBus\textsuperscript{®} (power management bus) compatible.

DMA can be used to reduce CPU overload.

22.2 I2C main features

- I\textsuperscript{2}C bus specification rev03 compatibility:
  - Slave and master modes
  - Multimaster capability
  - Standard-mode (up to 100 kHz)
  - Fast-mode (up to 400 kHz)
  - Fast-mode Plus (up to 1 MHz)
  - 7-bit and 10-bit addressing mode
  - Multiple 7-bit slave addresses (2 addresses, 1 with configurable mask)
  - All 7-bit addresses acknowledge mode
  - General call
  - Programmable setup and hold times
  - Easy to use event management
  - Optional clock stretching
  - Software reset
- 1-byte buffer with DMA capability
- Programmable analog and digital noise filters

The following features are also available, depending upon product implementation (see Section 22.3):

- SMBus specification rev 3.0 compatibility:
  - Hardware PEC (packet error checking) generation and verification with ACK control
  - Command and data acknowledge control
  - Address resolution protocol (ARP) support
  - Host and device support
  - SMBus alert
  - Timeouts and idle condition detection
- PMBus rev 1.3 standard compatibility
- Independent clock: a choice of independent clock sources allowing the I2C communication speed to be independent from the PCLK reprogramming
22.3 I2C implementation

This manual describes the full set of features implemented in I2C1. I2C2 supports a smaller set of features, but is otherwise identical to I2C1. The differences are listed below.

Table 68. STM32F0x0 I2C implementation

<table>
<thead>
<tr>
<th>I2C features(1)</th>
<th>STM32F030x4, STM32F030x6, STM32F070x6</th>
<th>STM32F030x8</th>
<th>STM32F070xB STM32F030xC</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C1</td>
<td>I2C1</td>
<td>I2C2</td>
<td>I2C1</td>
</tr>
<tr>
<td>7-bit addressing mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>10-bit addressing mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Standard mode (up to 100 kbit/s)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fast-mode (up to 400 kbit/s)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fast-mode Plus with 20 mA output drive I/Os (up to 1 Mbit/s)</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Independent clock</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Wake-up from Stop mode</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SMBus/PMBus</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

1. X = supported.

22.4 I2C functional description

In addition to receiving and transmitting data, this interface converts them from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the \( \text{I}^{2}\text{C} \) bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz), Fast-mode (up to 400 kHz) or Fast-mode Plus (up to 1 MHz) \( \text{I}^{2}\text{C} \) bus.

This interface can also be connected to an SMBus with data (SDA) and clock (SCL) pins. If the SMBus feature is supported, the optional SMBus Alert pin (SMBA) is also available.
22.4.1 I2C block diagram

The block diagram of the I2C interface is shown in Figure 199.

![Figure 199. I2C block diagram](image)

The I2C is clocked by an independent clock source, which allows the I2C to operate independently from the PCLK frequency.

For I2C I/Os supporting 20 mA output current drive for Fast-mode Plus operation, the driving capability is enabled through control bits in the system configuration controller (SYSCFG). Refer to Section 22.3: I2C implementation.

22.4.2 I2C2 block diagram

The block diagram of the I2C2 interface is shown in Figure 200.
22.4.3 I2C pins and internal signals

Table 69. I2C input/output pins

<table>
<thead>
<tr>
<th>Pin name</th>
<th>Signal type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_SDA</td>
<td>Bidirectional</td>
<td>I2C data</td>
</tr>
<tr>
<td>I2C_SCL</td>
<td>Bidirectional</td>
<td>I2C clock</td>
</tr>
<tr>
<td>I2C_SMBA</td>
<td>Bidirectional</td>
<td>SMBus alert</td>
</tr>
</tbody>
</table>

Table 70. I2C internal input/output signals

<table>
<thead>
<tr>
<th>Internal signal name</th>
<th>Signal type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_ker_ck</td>
<td>Input</td>
<td>I2C kernel clock, also named I2CCLK in this document</td>
</tr>
<tr>
<td>i2c_pclk</td>
<td>Input</td>
<td>I2C APB clock</td>
</tr>
<tr>
<td>i2c_it</td>
<td>Output</td>
<td>I2C interrupts, refer to Table 84 for the list of interrupt sources</td>
</tr>
<tr>
<td>i2c_rx_dma</td>
<td>Output</td>
<td>I2C receive data DMA request (I2C_RX)</td>
</tr>
<tr>
<td>i2c_tx_dma</td>
<td>Output</td>
<td>I2C transmit data DMA request (I2C_TX)</td>
</tr>
</tbody>
</table>

22.4.4 I2C clock requirements

The I2C kernel is clocked by I2CCLK.
The I2CCLK period $t_{\text{I2CCLK}}$ must respect the following conditions:

- $t_{\text{I2CCLK}} < (t_{\text{LOW}} - t_{\text{filters}}) / 4$
- $t_{\text{I2CCLK}} < t_{\text{HIGH}}$

with:

- $t_{\text{LOW}}$: SCL low time and $t_{\text{HIGH}}$: SCL high time
- $t_{\text{filters}}$: when enabled, sum of the delays brought by the analog and by the digital filters.

The digital filter delay is $\text{DNF} \times t_{\text{I2CCLK}}$.

The PCLK clock period $t_{\text{PCLK}}$ must respect the condition:

- $t_{\text{PCLK}} < 4 / 3 t_{\text{SCL}}$ ($t_{\text{SCL}}$: SCL period)

**Caution:** When the I2C kernel is clocked by PCLK, this clock must respect the conditions for $t_{\text{I2CCLK}}$.

### 22.4.5 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 when 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 I2C 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 can be enabled or disabled by software. The reserved SMBus addresses can be enabled also by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the START condition contains the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in master mode.

A ninth clock pulse follows the eight clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter (see *Figure 201*).
Acknowledge can be enabled or disabled by software. The I2C interface addresses can be selected by software.

### 22.4.6 I2C initialization

#### Enabling and disabling the peripheral

The I2C peripheral clock must be configured and enabled in the clock controller, then the I2C can be enabled by setting the PE bit in the I2C_CR1 register.

When the I2C is disabled (PE = 0), the I2C performs a software reset. Refer to Section 22.4.7 for more details.

#### Noise filters

Before enabling the I2C peripheral by setting the PE bit in I2C_CR1 register, the user must configure the noise filters, if needed. By default, an analog noise filter is present on the SDA and SCL inputs. This filter is compliant with the I2C specification, which requires the suppression of spikes with pulse width up to 50 ns in Fast-mode and Fast-mode Plus. The user can disable this analog filter by setting the ANFOFF bit, and/or select a digital filter by configuring the DNF[3:0] bit in the I2C_CR1 register.

When the digital filter is enabled, the level of the SCL or the SDA line is internally changed only if it remains stable for more than DNF x I2CCLK periods. This allows to suppress spikes with a programmable length of one to fifteen I2CCLK periods.

#### Table 71. Comparison of analog vs. digital filters

<table>
<thead>
<tr>
<th></th>
<th>Analog filter</th>
<th>Digital filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulse width of</td>
<td>≥ 50 ns</td>
<td>Programmable length, from one to fifteen I2C peripheral clocks</td>
</tr>
<tr>
<td>suppressed spikes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Caution:** The filter configuration cannot be changed when the I2C is enabled.

I2C timings
The timings must be configured to guarantee correct data hold and setup times, in master and slave modes. This is done by programming the PRESC[3:0], SCLDEL[3:0] and SDADEL[3:0] bits in the I2C_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C_TIMINGR content in the I2C configuration window.

Figure 202. Setup and hold timings

When the SCL falling edge is internally detected, a delay ($t_{SDADEL}$, impacting the hold time $t_{HD;DAT}$) is inserted before sending SDA output: $t_{SDADEL} = SDADEL \times t_{PRES} + t_{I2CCLK}$, where $t_{PRES} = (PRES + 1) \times t_{I2CCLK}$. 
The total SDA output delay is:
\[ t_{\text{SYNC1}} + \left\lfloor \text{SDADEL} \times (\text{PRESC} + 1) + 1 \right\rfloor \times t_{\text{I2CCLK}} \]

\( t_{\text{SYNC1}} \) duration depends upon:
- SCL falling slope
- When enabled, input delay brought by the analog filter: \( t_{\text{AF(min)}} < t_{\text{AF}} < t_{\text{AF(max)}} \)
- When enabled, input delay brought by the digital filter: \( t_{\text{DNF}} = \text{DNF} \times t_{\text{I2CCLK}} \)
- Delay due to SCL synchronization to I2CCLK clock (two to three I2CCLK periods)

To bridge the undefined region of the SCL falling edge, the user must program SDADEL in such a way that:

\[ \{t_{\text{f(max)}} + t_{\text{HD;DAT}}(\text{min}) - t_{\text{AF(min)}} - \left\lfloor \text{DNF} + 3 \right\rfloor \times t_{\text{I2CCLK}}\} / \{(\text{PRESC} + 1) \times t_{\text{I2CCLK}}\} \leq \text{SDADEL} \]

\[ \text{SDADEL} \leq \{t_{\text{HD;DAT}}(\text{max}) - t_{\text{AF(max)}} - \left\lfloor \text{DNF} + 4 \right\rfloor \times t_{\text{I2CCLK}}\} / \{(\text{PRESC} + 1) \times t_{\text{I2CCLK}}\} \]

**Note:** \( t_{\text{AF(min)}} / t_{\text{AF(max)}} \) are part of the equation only when the analog filter is enabled. Refer to the device datasheet for \( t_{\text{AF}} \) values.

The maximum \( t_{\text{HD;DAT}} \) can be 3.45 µs for Standard-mode, 0.9 µs for Fast-mode, 0.45 µs for Fast-mode Plus. It must be lower than the maximum of \( t_{\text{VD;DAT}} \) by a transition time. This maximum must only be met if the device does not stretch the LOW period (\( t_{\text{LOW}} \)) of the SCL signal. If the clock stretches the SCL, the data must be valid by the set-up time before it releases the clock.

The SDA rising edge is usually the worst case. In this case the previous equation becomes:

\[ \text{SDADEL} \leq \{t_{\text{VD;DAT}}(\text{max}) - t_{\text{r(max)}} - t_{\text{AF(max)}} - \left\lfloor \text{DNF} + 4 \right\rfloor \times t_{\text{I2CCLK}}\} / \{(\text{PRESC} + 1) \times t_{\text{I2CCLK}}\} \]

**Note:** This condition can be violated when NOSTRETCH = 0, because the device stretches SCL low to guarantee the set-up time, according to the SCLDEL value.

Refer to Table 72 for \( t_{\text{r}}, t_{\text{HD;DAT}}, \) and \( t_{\text{VD;DAT}} \) standard values.

- After \( t_{\text{SDADEL}} \), or after sending SDA output when the slave had to stretch the clock because the data was not yet written in I2C_TXDR register, SCL line is kept at low level during the setup time. This setup time is \( t_{\text{SCLDEL}} = (\text{SCLDEL} + 1) \times t_{\text{PRESC}}, \) where \( t_{\text{PRESC}} = (\text{PRESC} + 1) \times t_{\text{I2CCLK}} \). 

To bridge the undefined region of the SDA transition (rising edge usually worst case), the user must program SCLDEL in such a way that:

\[ \{t_{\text{r(max)}} + t_{\text{SU;DAT}}(\text{min})\} / \{(\text{PRESC} + 1) \times t_{\text{I2CCLK}}\} - 1 \leq \text{SCLDEL} \]

Refer to Table 72 for \( t_{\text{r}} \) and \( t_{\text{SU;DAT}} \) standard values.

The SDA and SCL transition time values to use are the ones in the application. Using the maximum values from the standard increases the constraints for the SDADEL and SCLDEL calculation, but ensures the feature, whatever the application.

**Note:** At every clock pulse, after SCL falling edge detection, the I2C master or slave stretches SCL low during at least \([\text{SDADEL} + \text{SCLDEL} + 1] \times (\text{PRESC} + 1) + 1 \times t_{\text{I2CCLK}}\). In both transmission and reception modes. In transmission mode, if the data is not yet written in I2C_TXDR when SDADEL counter is finished, the I2C keeps on stretching SCL low until the next data is written. Then new data MSB is sent on SDA output, and SCLDEL counter starts, continuing stretching SCL low to guarantee the data setup time.

If NOSTRETCH = 1 in slave mode, the SCL is not stretched, hence the SDADEL must be programmed so that it guarantees a sufficient setup time.
Additionally, in master mode, the SCL clock high and low levels must be configured by programming the PRESC[3:0], SCLH[7:0] and SCLL[7:0] bit fields in the I2C_TIMINGR register.

- When the SCL falling edge is internally detected, a delay is inserted before releasing the SCL output. This delay is \( t_{SCLL} = (SCLL + 1) \times t_{PRESC} \) where \( t_{PRESC} = (PRESC + 1) \times t_{I2CCLK} \). \( t_{SCLL} \) impacts the SCL low time \( t_{LOW} \).
- When the SCL rising edge is internally detected, a delay is inserted before forcing the SCL output to low level. This delay is \( t_{SCLH} = (SCLH + 1) \times t_{PRESC} \), where \( t_{PRESC} = (PRESC + 1) \times t_{I2CCLK} \). \( t_{SCLH} \) impacts the SCL high time \( t_{HIGH} \).

Refer to \textit{I2C master initialization} for more details.

**Caution:** Changing the timing configuration is not allowed when the I2C is enabled.

The I2C slave NOSTRETCH mode must also be configured before enabling the peripheral. Refer to \textit{I2C slave initialization} for more details.

**Caution:** Changing the NOSTRETCH configuration is not allowed when the I2C is enabled.

### Table 72. I2C-SMBus specification data setup and hold times

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Standard-mode (Sm)</th>
<th>Fast-mode (Fm)</th>
<th>Fast-mode Plus (Fm+)</th>
<th>SMBus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{HD,\text{DAT}} )</td>
<td>Data hold time</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>( t_{VD,\text{DAT}} )</td>
<td>Data valid time</td>
<td>-</td>
<td>3.45</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>( t_{SUL,\text{DAT}} )</td>
<td>Data setup time</td>
<td>250</td>
<td>-</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>( t_r )</td>
<td>Rise time of both SDA and SCL signals</td>
<td>-</td>
<td>1000</td>
<td>-</td>
<td>300</td>
</tr>
<tr>
<td>( t_f )</td>
<td>Fall time of both SDA and SCL signals</td>
<td>-</td>
<td>300</td>
<td>-</td>
<td>300</td>
</tr>
</tbody>
</table>
22.4.7 Software reset

A software reset can be performed by clearing the PE bit in the I2C_CR1 register. In that case I2C lines SCL and SDA are released. Internal states machines are reset and communication control bits, as well as status bits, come back to their reset value. The configuration registers are not impacted.

Impacted register bits:
1. I2C_CR2 register: START, STOP, NACK
2. I2C_ISR register: BUSY, TXE, TXIS, RXNE, ADDR, NACKF, TCR, TC, STOPF, BERR, ARLO, OVR

In addition when the SMBus feature is supported:
1. I2C_CR2 register: PECBYTE
2. I2C_ISR register: PECERR, TIMEOUT, ALERT

PE must be kept low during at least three APB clock cycles to perform the software reset. This is ensured by the following software sequence:
1. Write PE = 0
2. Check PE = 0
3. Write PE = 1
22.4.8 Data transfer

The data transfer is managed through transmit and receive data registers and a shift register.

Reception

The SDA input fills the shift register. After the eighth SCL pulse (when the complete data byte is received), the shift register is copied into I2C_RXDR register if it is empty (RXNE = 0). If RXNE = 1, meaning that the previous received data byte has not yet been read, the SCL line is stretched low until I2C_RXDR is read. The stretch is inserted between the eighth and ninth SCL pulse (before the acknowledge pulse).

Figure 204. Data reception
Transmission

If the I2C_TXDR register is not empty (TXE = 0), its content is copied into the shift register after the ninth SCL pulse (the acknowledge pulse). Then the shift register content is shifted out on SDA line. If TXE = 1, meaning that no data is written yet in I2C_TXDR, SCL line is stretched low until I2C_TXDR is written. The stretch is done after the ninth SCL pulse.

Figure 205. Data transmission

Hardware transfer management

The I2C features an embedded byte counter to manage byte transfer and to close the communication in various modes, such as:

- NACK, STOP and ReSTART generation in master mode
- ACK control in slave receiver mode
- PEC generation/checking when SMBus feature is supported

The byte counter is always used in master mode. By default, it is disabled in slave mode. It can be enabled by software by setting the SBC (slave byte control) bit in the I2C_CR1 register.

The number of bytes to be transferred is programmed in the NBYTES[7:0] bit field in the I2C_CR2 register. If the number of bytes to be transferred (NBYTES) is greater than 255, or if a receiver wants to control the acknowledge value of a received data byte, the reload mode must be selected by setting the RELOAD bit in the I2C_CR2 register. In this mode, the TCR flag is set when the number of bytes programmed in NBYTES is transferred, and an interrupt is generated if TCIE is set. SCL is stretched as long as TCR flag is set. TCR is cleared by software when NBYTES is written to a non-zero value.

When the NBYTES counter is reloaded with the last number of bytes, RELOAD bit must be cleared.
When RELOAD = 0 in master mode, the counter can be used in two modes:

- **Automatic end** (AUTOEND = 1 in the I2C_CR2 register). In this mode, the master automatically sends a STOP condition once the number of bytes programmed in the NBYTES[7:0] bit field is transferred.

- **Software end** (AUTOEND = 0 in the I2C_CR2 register). In this mode, software action is expected once the number of bytes programmed in the NBYTES[7:0] bit field is transferred; the TC flag is set and an interrupt is generated if the TCIE bit is set. The SCL signal is stretched as long as the TC flag is set. The TC flag is cleared by software when the START or STOP bit is set in the I2C_CR2 register. This mode must be used when the master wants to send a RESTART condition.

**Caution:** The AUTOEND bit has no effect when the RELOAD bit is set.

### Table 73. I2C configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>SBC bit</th>
<th>RELOAD bit</th>
<th>AUTOEND bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Tx/Rx NBYTES + STOP</td>
<td>x</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Master Tx/Rx + NBYTES + RESTART</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Slave Tx/Rx, all received bytes ACKed</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Slave Rx with ACK control</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

#### 22.4.9 I2C slave mode

**I2C slave initialization**

To work in slave mode, the user must enable at least one slave address. Registers I2C_OAR1 and I2C_OAR2 are available to program the slave own addresses OA1 and OA2.

- OA1 can be configured either in 7-bit mode (by default), or in 10-bit addressing mode by setting the OA1MODE bit in the I2C_OAR1 register.
  - OA1 is enabled by setting the OA1EN bit in the I2C_OAR1 register.

- If additional slave addresses are required, the second slave address OA2 can be configured. Up to seven OA2 LSB can be masked by configuring the OA2MSK[2:0] bits in the I2C_OAR2 register. Therefore for OA2MSK configured from 1 to 6, only OA2[7:2], OA2[7:3], OA2[7:4], OA2[7:5], OA2[7:6] or OA2[7] are compared with the received address. As soon as OA2MSK is not equal to 0, the address comparator for OA2 excludes the I2C reserved addresses (0000 XXX and 1111 XXX), which are not acknowledged. If OA2MSK = 7, all received 7-bit addresses are acknowledged (except reserved addresses). OA2 is always a 7-bit address.
  - These reserved addresses can be acknowledged if they are enabled by the specific enable bit, if they are programmed in the I2C_OAR1 or I2C_OAR2 register with OA2MSK = 0.
  - OA2 is enabled by setting the OA2EN bit in the I2C_OAR2 register.

- The general call address is enabled by setting the GCEN bit in the I2C_CR1 register.

When the I2C is selected by one of its enabled addresses, the ADDR interrupt status flag is set, and an interrupt is generated if the ADDRIE bit is set.

By default, the slave uses its clock stretching capability, which means that it stretches the SCL signal at low level when needed, to perform software actions. If the master does not
support clock stretching, the I2C must be configured with NOSTRETCH = 1 in the I2C_CR1 register.

After receiving an ADDR interrupt, if several addresses are enabled, the user must read the ADDCODE[6:0] bits in the I2C_ISR register to check which address matched. DIR flag must also be checked to know the transfer direction.

**Slave clock stretching (NOSTRETCH = 0)**

In default mode, the I2C slave stretches the SCL clock in the following situations:

- When the ADDR flag is set: the received address matches with one of the enabled slave addresses. This stretch is released when the ADDR flag is cleared by software setting the ADDRCF bit.
- In transmission, if the previous data transmission is completed and no new data is written in I2C_TXDR register, or if the first data byte is not written when the ADDR flag is cleared (TXE = 1). This stretch is released when the data is written to the I2C_TXDR register.
- In reception when the I2C_RXDR register is not read yet and a new data reception is completed. This stretch is released when I2C_RXDR is read.
- When TCR = 1 in Slave byte control mode, reload mode (SBC = 1 and RELOAD = 1), meaning that the last data byte has been transferred. This stretch is released when then TCR is cleared by writing a non-zero value in the NBYTES[7:0] field.
- After SCL falling edge detection, the I2C stretches SCL low during \([(SDADEL + SCLDEL + 1) x (PRESCL + 1)] x t_{I2CCLK}\).

**Slave without clock stretching (NOSTRETCH = 1)**

When NOSTRETCH = 1 in the I2C_CR1 register, the I2C slave does not stretch the SCL signal.

- The SCL clock is not stretched while the ADDR flag is set.
- In transmission, the data must be written in the I2C_TXDR register before the first SCL pulse corresponding to its transfer occurs. If not, an underrun occurs, the OVR flag is set in the I2C_ISR register and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register. The OVR flag is also set when the first data transmission starts and the STOPF bit is still set (has not been cleared). Therefore, if the user clears the STOPF flag of the previous transfer only after writing the first data to be transmitted in the next transfer, it ensures that the OVR status is provided, even for the first data to be transmitted.
- In reception, the data must be read from the I2C_RXDR register before the ninth SCL pulse (ACK pulse) of the next data byte occurs. If not, an overrun occurs, the OVR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.
Slave byte control mode

To allow byte ACK control in slave reception mode, the Slave byte control mode must be enabled by setting the SBC bit in the I2C_CR1 register. This is required to be compliant with SMBus standards.

The Reload mode must be selected to allow byte ACK control in slave reception mode (RELOAD = 1). To get control of each byte, NBYTES must be initialized to 0x1 in the ADDR interrupt subroutine, and reloaded to 0x1 after each received byte. When the byte is received, the TCR bit is set, stretching the SCL signal low between the eighth and ninth SCL pulses. The user can read the data from the I2C_RXDR register, and then decide to acknowledge it or not by configuring the ACK bit in the I2C_CR2 register. The SCL stretch is released by programming NBYTES to a non-zero value: the acknowledge or not-acknowledge is sent, and the next byte can be received.

NBYTES can be loaded with a value greater than 0x1, and in this case, the reception flow is continuous during NBYTES data reception.

Note: The SBC bit must be configured when the I2C is disabled, or when the slave is not addressed, or when ADDR = 1.

The RELOAD bit value can be changed when ADDR = 1, or when TCR = 1.

Caution: The Slave byte control mode is not compatible with NOSTRETCH mode. Setting SBC when NOSTRETCH = 1 is not allowed.

Figure 206. Slave initialization flow

1. SBC must be set to support SMBus features.

For a code example refer to A.11.3: I2C configured in slave mode.
Slave transmitter

A transmit interrupt status (TXIS) is generated when the I2C_TXDR register becomes empty. An interrupt is generated if the TXIE bit is set in the I2C_CR1 register.

The TXIS bit is cleared when the I2C_TXDR register is written with the next data byte to be transmitted.

When a NACK is received, the NACKF bit is set in the I2C_ISR register, and an interrupt is generated if the NACKIE bit is set in the I2C_CR1 register. The slave automatically releases the SCL and SDA lines to let the master perform a STOP or a RESTART condition. The TXIS bit is not set when a NACK is received.

When a STOP is received and the STOPIE bit is set in the I2C_CR1 register, the STOPF flag is set in the I2C_ISR register and an interrupt is generated. In most applications, the SBC bit is usually programmed to 0. In this case, if TXE = 0 when the slave address is received (ADDR = 1), the user can choose either to send the content of the I2C_TXDR register as the first data byte, or to flush the I2C_TXDR register by setting the TXE bit in order to program a new data byte.

In Slave byte control mode (SBC = 1), the number of bytes to be transmitted must be programmed in NBYTES in the address match interrupt subroutine (ADDR = 1). In this case, the number of TXIS events during the transfer corresponds to the value programmed in NBYTES.

Caution: When NOSTRETCH = 1, the SCL clock is not stretched while the ADDR flag is set, so the user cannot flush the I2C_TXDR register content in the ADDR subroutine, to program the first data byte. The first data byte to be sent must be previously programmed in the I2C_TXDR register:

- This data can be the one written in the last TXIS event of the previous transmission message.
- If this data byte is not the one to be sent, the I2C_TXDR register can be flushed by setting the TXE bit in order to program a new data byte. The STOPF bit must be cleared only after these actions, in order to guarantee that they are executed before the first data transmission starts, following the address acknowledge.

If STOPF is still set when the first data transmission starts, an underrun error is generated (the OVR flag is set).

If a TXIS event (transmit interrupt or transmit DMA request) is needed, the user must set the TXIS bit in addition to the TXE bit, to generate the event.
Slave initialization

Read ADDCODE and DIR in I2C_ISR
Optional: Set I2C_ISR.TXE = 1
Set I2C_ICR.ADDRCF

Write I2C_TXDR.TXDATA
Figure 208. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 1

Slave initialization

I2C_ISR.TXIS = 1?

Yes

Write I2C_TXDR.TXDATA

No

I2C_ISR.TXIS = 1?

Yes

Set I2C_ICR.STOPCF

No

I2C_ISR.STOPF = 1?

Yes

Optional: Set I2C_ISR.TXE = 1 and I2C_ISR.TXIS=1

No
Figure 209. Transfer bus diagrams for I2C slave transmitter (mandatory events only)

Example I2C slave transmitter 3 bytes with 1st data flushed, NOSTRETCH=0:

EV1: ADDR ISR: check ADDCODE and DIR, set TXE, set ADDRCF
EV2: TXIS ISR: wr data1
EV3: TXIS ISR: wr data2
EV4: TXIS ISR: wr data3
EV5: TXIS ISR: wr data4 (not sent)

Example I2C slave transmitter 3 bytes without 1st data flush, NOSTRETCH=0:

EV1: ADDR ISR: check ADDCODE and DIR, set ADDRCF
EV2: TXIS ISR: wr data2
EV3: TXIS ISR: wr data3
EV4: TXIS ISR: wr data4 (not sent)

Example I2C slave transmitter 3 bytes, NOSTRETCH=1:

EV1: wr data1
EV2: TXIS ISR: wr data2
EV3: TXIS ISR: wr data3
EV4: TXIS ISR: wr data4 (not sent)
EV5: STOPF ISR: (optional: set TXE and TXIS), set STOPCF

For a code example refer to A.11.6: I2C slave transmitter.
Slave receiver

RXNE is set in I2C_ISR when the I2C_RXDR is full, and generates an interrupt if RXIE is set in I2C_CR1. RXNE is cleared when I2C_RXDR is read.

When a STOP is received and STOPIE is set in I2C_CR1, STOPF is set in I2C_ISR and an interrupt is generated.

Figure 210. Transfer sequence flow for slave receiver with NOSTRETCH = 0

```
Slave reception

Slave initialization

I2C_ISR.ADDR =1?

No

Read ADDCODE and DIR in I2C_ISR
Set I2C_ICR.ADDRCF

Yes

I2C_ISR.RXNE =1?

No

Write I2C_RXDR.RXDATA

Yes

SCL stretched
```
Figure 211. Transfer sequence flow for slave receiver with NOSTRETCH = 1

Slave reception

Slave initialization

\[ \text{I2C_ISR.RXNE} = \text{1?} \]

Yes

Read I2C_RXDR.RXDATA

No

\[ \text{I2C_ISR.STOPF} = \text{1?} \]

Yes

Set I2C_ICR.STOPCF

No

Figure 212. Transfer bus diagrams for I2C slave receiver (mandatory events only)

Example I2C slave receiver 3 bytes, NOSTRETCH = 0:

<table>
<thead>
<tr>
<th>ADDR</th>
<th>RXNE</th>
<th>RXNE</th>
<th>RXNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EV1: ADDR ISR: check ADDCODE and DIR, set ADDRCF
EV2: RXNE ISR: rd data1
EV3: RXNE ISR: rd data2
EV4: RXNE ISR: rd data3

Example I2C slave receiver 3 bytes, NOSTRETCH = 1:

<table>
<thead>
<tr>
<th>ADDR</th>
<th>RXNE</th>
<th>RXNE</th>
<th>RXNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EV1: RXNE ISR: rd data1
EV2: RXNE ISR: rd data2
EV3: RXNE ISR: rd data3

For a code example refer to A.11.7: I2C slave receiver.
22.4.10 I2C master mode

I2C master initialization

Before enabling the peripheral, the I2C master clock must be configured by setting the SCLH and SCLL bits in the I2C_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C_TIMINGR content in the I2C Configuration window.

A clock synchronization mechanism is implemented in order to support multi-master environment and slave clock stretching.

In order to allow clock synchronization:

- The low level of the clock is counted using the SCLL counter, starting from the SCL low level internal detection.
- The high level of the clock is counted using the SCLH counter, starting from the SCL high level internal detection.

The I2C detects its own SCL low level after a $t_{SYNC1}$ delay depending on the SCL falling edge, SCL input noise filters (analog and digital), and SCL synchronization to the I2CxCLK clock. The I2C releases SCL to high level once the SCLL counter reaches the value programmed in the SCLL[7:0] bits in the I2C_TIMINGR register.

The I2C detects its own SCL high level after a $t_{SYNC2}$ delay depending on the SCL rising edge, SCL input noise filters (analog + digital) and SCL synchronization to I2CxCLK clock. The I2C ties SCL to low level once the SCLH counter reaches the value programmed in the SCLH[7:0] bits in the I2C_TIMINGR register.

Consequently the master clock period is:

$$t_{SCL} = t_{SYNC1} + t_{SYNC2} + \{(SCLH+ 1) + (SCLL+ 1)\} \times (PRESC+ 1) \times t_{I2CCLK}$$

The duration of $t_{SYNC1}$ depends upon:

- SCL falling slope
- When enabled, input delay induced by the analog filter
- When enabled, input delay induced by the digital filter: DNF $\times t_{I2CCLK}$
- Delay due to SCL synchronization with I2CCLK clock (two to three I2CCLK periods)

The duration of $t_{SYNC2}$ depends upon:

- SCL rising slope
- When enabled, input delay induced by the analog filter
- When enabled, input delay induced by the digital filter: DNF $\times t_{I2CCLK}$
- Delay due to SCL synchronization with I2CCLK clock (two to three I2CCLK periods)
Figure 213. Master clock generation

SCL master clock generation

- SCL high level detected
- SCLH counter starts
- SCL released
- SCL driven low
- tSYNC1
- SCL low level detected
- SCLL counter starts
- SCL driven low
- tSYNC2
- SCL high level detected
- SCLH counter starts
- SCL low level detected
- SCLL counter starts
- SCL driven low by another device
- SCL driven low by another device
- SCL low level detected
- SCLL counter starts
- SCL released

SCL master clock synchronization

- SCL high level detected
- SCLH counter starts
- SCL driven low by another device
- SCL low level detected
- SCLL counter starts
- SCL driven low by another device
- SCL low level detected
- SCLL counter starts
- SCL released
Caution: To be I2C or SMBus compliant, the master clock must respect the timings given in the following table.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Standard-mode (Sm)</th>
<th>Fast-mode (Fm)</th>
<th>Fast-mode Plus (Fm+)</th>
<th>SMBus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>$f_{SCL}$</td>
<td>SCL clock frequency</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>400</td>
</tr>
<tr>
<td>$t_{HD:STA}$</td>
<td>Hold time (repeated) START condition</td>
<td>4.0</td>
<td>-</td>
<td>0.6</td>
<td>-</td>
</tr>
<tr>
<td>$t_{SU:STA}$</td>
<td>Set-up time for a repeated START condition</td>
<td>4.7</td>
<td>-</td>
<td>0.6</td>
<td>-</td>
</tr>
<tr>
<td>$t_{SU:STO}$</td>
<td>Set-up time for STOP condition</td>
<td>4.0</td>
<td>-</td>
<td>0.6</td>
<td>-</td>
</tr>
<tr>
<td>$t_{BUF}$</td>
<td>Bus free time between a STOP and START condition</td>
<td>4.7</td>
<td>-</td>
<td>1.3</td>
<td>-</td>
</tr>
<tr>
<td>$t_{LOW}$</td>
<td>Low period of the SCL clock</td>
<td>4.7</td>
<td>-</td>
<td>1.3</td>
<td>-</td>
</tr>
<tr>
<td>$t_{HIGH}$</td>
<td>Period of the SCL clock</td>
<td>4.0</td>
<td>-</td>
<td>0.6</td>
<td>-</td>
</tr>
<tr>
<td>$t_{r}$</td>
<td>Rise time of both SDA and SCL signals</td>
<td>-</td>
<td>1000</td>
<td>-</td>
<td>300</td>
</tr>
<tr>
<td>$t_{f}$</td>
<td>Fall time of both SDA and SCL signals</td>
<td>-</td>
<td>300</td>
<td>-</td>
<td>300</td>
</tr>
</tbody>
</table>

Table 74. I2C-SMBus specification clock timings

Note: $SCLL$ and $SCLH$ are also used to generate, respectively, the $t_{BUF}$/$t_{SU:STA}$ and the $t_{HD:STA}$/$t_{SU:STO}$ timings.

Refer to Section 22.4.11 for examples of I2C_TIMINGR settings vs. I2CCLK frequency.

Master communication initialization (address phase)

To initiate the communication, program the following parameters for the addressed slave in the I2C_CR2 register:

- Addressing mode (7-bit or 10-bit): ADD10
- Slave address to be sent: SADD[9:0]
- Transfer direction: RD_WRN
- In case of 10-bit address read: HEAD10R bit. HEAD10R must be configure to indicate if the complete address sequence must be sent, or only the header in case of a direction change.
- The number of bytes to be transferred: NBYTES[7:0]. If this number is equal to or greater than 255 bytes, NBYTES[7:0] must initially be filled with 0xFF.

The user must then set the START bit in I2C_CR2 register. Changing all the above bits is not allowed when START bit is set.

Then the master automatically sends the START condition followed by the slave address as soon as it detects that the bus is free (BUSY = 0) and after a $t_{BUF}$ delay.

In case of an arbitration loss, the master automatically switches back to slave mode and can acknowledge its own address if it is addressed as a slave.
Note: The START bit is reset by hardware when the slave address is sent on the bus, whatever the received acknowledge value. The START bit is also reset by hardware if an arbitration loss occurs.

In 10-bit addressing mode, when the slave address first seven bits are NACKed by the slave, the master relaunches automatically the slave address transmission until ACK is received. In this case ADDRCF must be set if a NACK is received from the slave, to stop sending the slave address.

If the I2C is addressed as a slave (ADDR = 1) while the START bit is set, the I2C switches to slave mode, and the START bit is cleared, when the ADDRCF bit is set.

Note: The same procedure is applied for a repeated start condition. In this case BUSY = 1.

For code examples refer to A.11.1: I2C configured in master mode to receive and A.11.2: I2C configured in master mode to transmit.

Initialization of a master receiver addressing a 10-bit address slave

- If the slave address is in 10-bit format, the user can choose to send the complete read sequence by clearing the HEAD10R bit in the I2C_CR2 register. In this case the master automatically sends the following complete sequence after the START bit is set:
  (Re)Start + Slave address 10-bit header Write + Slave address second byte + (Re)Start + Slave address 10-bit header Read

Figure 215. 10-bit address read access with HEAD10R = 0
If the master addresses a 10-bit address slave, transmits data to this slave and then reads data from the same slave, a master transmission flow must be done first. Then a repeated start is set with the 10-bit slave address configured with HEAD10R = 1. In this case the master sends this sequence: ReStart + Slave address 10-bit header Read.

Figure 216. 10-bit address read access with HEAD10R = 1

Master transmitter

In the case of a write transfer, the TXIS flag is set after each byte transmission, after the ninth SCL pulse when an ACK is received.

A TXIS event generates an interrupt if the TXIE bit is set in the I2C_CR1 register. The flag is cleared when the I2C_TXDR register is written with the next data byte to be transmitted.

The number of TXIS events during the transfer corresponds to the value programmed in NBYTES[7:0]. If the total number of data bytes to be sent is greater than 255, reload mode must be selected by setting the RELOAD bit in the I2C_CR2 register. In this case, when NBYTES data have been transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written to a non-zero value.

The TXIS flag is not set when a NACK is received.

When RELOAD = 0 and NBYTES data have been transferred:
- In automatic end mode (AUTOEND = 1), a STOP is automatically sent.
- In software end mode (AUTOEND = 0), the TC flag is set and the SCL line is stretched low, to perform software actions:
  A RESTART condition can be requested by setting the START bit in the I2C_CR2 register with the proper slave address configuration, and number of bytes to be transferred. Setting the START bit clears the TC flag and the START condition is sent on the bus.
  A STOP condition can be requested by setting the STOP bit in the I2C_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.

If a NACK is received: the TXIS flag is not set, and a STOP condition is automatically sent after the NACK reception. the NACKF flag is set in the I2C_ISR register, and an interrupt is generated if the NACKIE bit is set.
Figure 217. Transfer sequence flow for I2C master transmitter for \(N \leq 255\) bytes

1. **Master transmission**
   - **Master initialization**
     - NBYTES = N
     - AUTOEND = 0 for RESTART, 1 for STOP
     - Configure slave address
     - Set I2C_CR2.START

2. **No**
   - I2C_ISR.NACKF = 1?
     - Yes
     - End
     - No
   - I2C_ISR.TXIS = 1?
     - Yes
     - Write I2C_TXDR
     - No
     - NBYTES transmitted?
       - Yes
       - I2C_ISR.TC = 1?
         - Yes
         - Set I2C_CR2.START with slave address NBYTES ...
         - No
         - End
       - No
         - End
Figure 218. Transfer sequence flow for I2C master transmitter for N > 255 bytes

1. Master initialization
   - NBYTES = 0xFF; N = N-255
   - RELOAD = 1
   - Configure slave address
   - Set I2C_CR2.START

2. Master transmission
   - I2C_ISR.NACKF = 1?
     - No
     - Yes

3. Write I2C_TXDR
   - I2C_ISR.TXS = 1?
     - No
     - Yes

4. NBYTES transmitted ?
   - No
   - Yes

5. Set I2C_CR2.START with slave address
   - NBYTES  ...

6. I2C_ISR.TCR = 1?
   - No
   - Yes

7. IF N <= 256
   - NBYTES = N; N = 0; RELOAD = 0
   - AUTOEND = 0 for RESTART; 1 for STOP
   - ELSE
     - NBYTES = 0xFF; N = N-255
     - RELOAD = 1

End
Figure 219. Transfer bus diagrams for I2C master transmitter
(mandatory events only)

Example I2C master transmitter 2 bytes, automatic end mode (STOP)

INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START
EV1: TXIS ISR: wr data1
EV2: TXIS ISR: wr data2

Example I2C master transmitter 2 bytes, software end mode (RESTART)

INIT: program Slave address, program NBYTES = 2, AUTOEND=0, set START
EV1: TXIS ISR: wr data1
EV2: TXIS ISR: wr data2
EV3: TC ISR: programSlave address, program NBYTES = N, set START

For a code example refer to A.11.4: I2C master transmitter.
Master receiver

In the case of a read transfer, the RXNE flag is set after each byte reception, after the eighth SCL pulse. An RXNE event generates an interrupt if the RXIE bit is set in the I2C_CR1 register. The flag is cleared when I2C_RXDR is read.

If the total number of data bytes to be received is greater than 255, reload mode must be selected by setting the RELOAD bit in the I2C_CR2 register. In this case, when NBYTES[7:0] data have been transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written to a non-zero value.

- When RELOAD = 0 and NBYTES[7:0] data have been transferred:
  - In automatic end mode (AUTOEND = 1), a NACK and a STOP are automatically sent after the last received byte.
  - In software end mode (AUTOEND = 0), a NACK is automatically sent after the last received byte, the TC flag is set and the SCL line is stretched low in order to allow software actions:
    A RESTART condition can be requested by setting the START bit in the I2C_CR2 register with the proper slave address configuration, and number of bytes to be transferred. Setting the START bit clears the TC flag and the START condition, followed by slave address, are sent on the bus.
    A STOP condition can be requested by setting the STOP bit in the I2C_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.
Figure 220. Transfer sequence flow for I2C master receiver for \( N \leq 255 \) bytes

- Master initialization

\[
\text{NBYTES} = N \\
\text{AUTOEND} = 0 \text{ for RESTART, 1 for STOP} \\
\text{Configure slave address} \\
\text{Set } \text{i2C\_CR2\_START}
\]

- \( \text{I2C\_ISR\_RXNE} = 1? \)
  - No
  - Yes
  - Read \text{I2C\_RXDR}

- \( \text{NBYTES received?} \)
  - Yes
  - No

- \( \text{I2C\_ISR\_TC} = 1? \)
  - No
  - Yes
  - Set \text{i2C\_CR2\_START} with slave address \text{NBYTES} ...
Figure 221. Transfer sequence flow for I2C master receiver for N > 255 bytes

Master initialization

NBYTES = 0xFF; N=N-255
RELOAD =1
Configure slave address
Set I2C_CR2.START

I2C_ISR.RXNE =1?
No
Yes
Read I2C_RXDR

NBYTES received?
Yes
No

I2C_ISR.TC =1?
No
Yes

I2C_ISR.TCR = 1?
No
Yes

IF N< 256
NBYTES =N; N=0;RELOAD=0
AUTOEND=0 for RESTART; 1 for STOP
ELSE
NBYTES =0xFF;N=N-255
RELOAD=1

End

Set I2C_CR2.START with slave address NBYTES ...
22.4.11 **I2C_TIMINGR register configuration examples**

The following tables provide examples of how to program the I2C_TIMINGR to obtain timings compliant with the I2C specification. To get more accurate configuration values, use the STM32CubeMX tool (I2C Configuration window).

For a code example refer to A.11.5: I2C master receiver.
### Table 75. Examples of timing settings for \(f_{I^2CCLK} = 8 \text{ MHz}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Standard-mode (Sm)</th>
<th>Fast-mode (Fm)</th>
<th>Fast-mode Plus (Fm+)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10 kHz</td>
<td>100 kHz</td>
<td>400 kHz</td>
</tr>
<tr>
<td>PRESC</td>
<td>0x1</td>
<td>0x1</td>
<td>0x0</td>
</tr>
<tr>
<td>SCLL</td>
<td>0xC7</td>
<td>0x13</td>
<td>0x9</td>
</tr>
<tr>
<td>(t_{SCLL})</td>
<td>200 x 250 ns = 50 (\mu)s</td>
<td>20 x 250 ns = 5.0 (\mu)s</td>
<td>10 x 125 ns = 1250 ns</td>
</tr>
<tr>
<td>SCLH</td>
<td>0xC3</td>
<td>0xF</td>
<td>0x3</td>
</tr>
<tr>
<td>(t_{SCLH})</td>
<td>196 x 250 ns = 49 (\mu)s</td>
<td>16 x 250 ns = 4.0 (\mu)s</td>
<td>4 x 125 ns = 500 ns</td>
</tr>
<tr>
<td>(t_{SCL}(1))</td>
<td>(~100 \mu)s(2)</td>
<td>(~10 \mu)s(2)</td>
<td>(~2.5 \mu)s(3)</td>
</tr>
<tr>
<td>SDADEL</td>
<td>0x2</td>
<td>0x2</td>
<td>0x1</td>
</tr>
<tr>
<td>(t_{SDADEL})</td>
<td>2 x 250 ns = 500 ns</td>
<td>2 x 250 ns = 500 ns</td>
<td>1 x 125 ns = 125 ns</td>
</tr>
<tr>
<td>SCLDEL</td>
<td>0x4</td>
<td>0x4</td>
<td>0x3</td>
</tr>
<tr>
<td>(t_{SCLDEL})</td>
<td>5 x 250 ns = 1250 ns</td>
<td>5 x 250 ns = 1250 ns</td>
<td>4 x 125 ns = 500 ns</td>
</tr>
</tbody>
</table>

1. \(t_{SCL}\) is greater than \(t_{SCLL} + t_{SCLH}\) due to SCL internal detection delay. Values provided for \(t_{SCL}\) are examples only.
2. \(t_{SYNC1} + t_{SYNC2}\) minimum value is \(4 \times t_{I^2CCLK} = 500\) ns. Example with \(t_{SYNC1} + t_{SYNC2} = 1000\) ns.
3. \(t_{SYNC1} + t_{SYNC2}\) minimum value is \(4 \times t_{I^2CCLK} = 600\) ns. Example with \(t_{SYNC1} + t_{SYNC2} = 750\) ns.
4. \(t_{SYNC1} + t_{SYNC2}\) minimum value is \(4 \times t_{I^2CCLK} = 655\) ns.

### Table 76. Examples of timing settings for \(f_{I^2CCLK} = 16 \text{ MHz}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Standard-mode (Sm)</th>
<th>Fast-mode (Fm)</th>
<th>Fast-mode Plus (Fm+)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10 kHz</td>
<td>100 kHz</td>
<td>400 kHz</td>
</tr>
<tr>
<td>PRESC</td>
<td>0x3</td>
<td>0x3</td>
<td>0x1</td>
</tr>
<tr>
<td>SCLL</td>
<td>0xC7</td>
<td>0x13</td>
<td>0x9</td>
</tr>
<tr>
<td>(t_{SCLL})</td>
<td>200 x 250 ns = 50 (\mu)s</td>
<td>20 x 250 ns = 5.0 (\mu)s</td>
<td>10 x 125 ns = 1250 ns</td>
</tr>
<tr>
<td>SCLH</td>
<td>0xC3</td>
<td>0xF</td>
<td>0x3</td>
</tr>
<tr>
<td>(t_{SCLH})</td>
<td>196 x 250 ns = 49 (\mu)s</td>
<td>16 x 250 ns = 4.0 (\mu)s</td>
<td>4 x 125 ns = 500 ns</td>
</tr>
<tr>
<td>(t_{SCL}(1))</td>
<td>(~100 \mu)s(2)</td>
<td>(~10 \mu)s(2)</td>
<td>(~2.5 \mu)s(3)</td>
</tr>
<tr>
<td>SDADEL</td>
<td>0x2</td>
<td>0x2</td>
<td>0x2</td>
</tr>
<tr>
<td>(t_{SDADEL})</td>
<td>2 x 250 ns = 500 ns</td>
<td>2 x 250 ns = 500 ns</td>
<td>2 x 125 ns = 250 ns</td>
</tr>
<tr>
<td>SCLDEL</td>
<td>0x4</td>
<td>0x4</td>
<td>0x3</td>
</tr>
<tr>
<td>(t_{SCLDEL})</td>
<td>5 x 250 ns = 1250 ns</td>
<td>5 x 250 ns = 1250 ns</td>
<td>4 x 125 ns = 500 ns</td>
</tr>
</tbody>
</table>

1. \(t_{SCL}\) is greater than \(t_{SCLL} + t_{SCLH}\) due to SCL internal detection delay. Values provided for \(t_{SCL}\) are examples only.
2. \(t_{SYNC1} + t_{SYNC2}\) minimum value is \(4 \times t_{I^2CCLK} = 250\) ns. Example with \(t_{SYNC1} + t_{SYNC2} = 1000\) ns.
3. \(t_{SYNC1} + t_{SYNC2}\) minimum value is \(4 \times t_{I^2CCLK} = 250\) ns. Example with \(t_{SYNC1} + t_{SYNC2} = 750\) ns.
4. \(t_{SYNC1} + t_{SYNC2}\) minimum value is \(4 \times t_{I^2CCLK} = 250\) ns. Example with \(t_{SYNC1} + t_{SYNC2} = 500\) ns.
22.4.12 SMBus specific features

This section is relevant only when the SMBus feature is supported (refer to Section 22.3).

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 I²C principles of operation. The SMBus provides a control bus for system and power management related tasks.

This peripheral is compatible with the SMBus specification (http://smbus.org).

The system management bus specification refers to three types of devices

- A slave is a device that receives or responds 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.

This peripheral can be configured as master or slave device, and also as a host.

Bus protocols

There are eleven possible command protocols for any given device. A device can use any or all of them to communicate. The protocols are Quick Command, Send Byte, Receive Byte, Write Byte, Write Word, Read Byte, Read Word, Process Call, Block Read, Block...
Write, and Block Write-Block Read Process Call. These protocols must be implemented by the user software.

For more details on these protocols, refer to SMBus specification (http://smbus.org).

**Address resolution protocol (ARP)**

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. To provide a mechanism to isolate each device for the purpose of address assignment, each device must implement a unique device identifier (UDID). This 128-bit number is implemented by software.

This peripheral supports the Address Resolution Protocol (ARP). The SMBus Device Default Address (0b1100 001) is enabled by setting SMBDEN bit in I2C_CR1 register. The ARP commands must be implemented by the user software.

Arbitration is also performed in slave mode for ARP support.

For more details of the SMBus address resolution protocol, refer to SMBus specification (http://smbus.org).

**Received command and data acknowledge control**

A SMBus receiver must be able to NACK each received command or data. In order to allow the ACK control in slave mode, the Slave byte control mode must be enabled by setting SBC bit in I2C_CR1 register. Refer to [Slave byte control mode](#) for more details.

**Host notify protocol**

This peripheral supports the host notify protocol by setting the SMBHEN bit in the I2C_CR1 register. In this case the host acknowledges the SMBus host address (0b0001 000).

When this protocol is used, the device acts as a master and the host as a slave.

**SMBus alert**

The SMBus ALERT optional signal is supported. A slave-only device can signal the host through the SMBALERT# pin that it wants to talk. The host processes the interrupt and simultaneously accesses all SMBALERT# devices through the alert response address (0b0001 100). Only the device(s) which pulled SMBALERT# low acknowledges the alert response address.

When configured as a slave device (SMBHEN = 0), the SMBA pin is pulled low by setting the ALERTEN bit in the I2C_CR1 register. The Alert Response Address is enabled at the same time.

When configured as a host (SMBHEN = 1), the ALERT flag is set in the I2C_ISR register when a falling edge is detected on the SMBA pin and ALERTEN = 1. An interrupt is generated if the ERRIE bit is set in the I2C_CR1 register. When ALERTEN = 0, the ALERT line is considered high even if the external SMBA pin is low.

*If the SMBus ALERT pin is not needed, the SMBA pin can be used as a standard GPIO if ALERTEN = 0.*

**Packet error checking**

A packet error checking mechanism has been introduced in the SMBus specification to improve reliability and communication robustness. The packet error checking is implemented by appending a packet error code (PEC) at the end of each message transfer.
The PEC is calculated by using the $C(x) = x^8 + x^2 + x + 1$ CRC-8 polynomial on all the message bytes (including addresses and read/write bits).

The peripheral embeds a hardware PEC calculator and allows a not acknowledge to be sent automatically when the received byte does not match with the hardware calculated PEC.

**Timeouts**

This peripheral embeds hardware timers to be compliant with the three timeouts defined in the SMBus specification.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{\text{TIMEOUT}}$</td>
<td>Detect clock low timeout</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>$t_{\text{LOW:SEXT}}^{(1)}$</td>
<td>Cumulative clock low extend time (slave device)</td>
<td>-</td>
<td>25</td>
</tr>
<tr>
<td>$t_{\text{LOW:MEXT}}^{(2)}$</td>
<td>Cumulative clock low extend time (master device)</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>

1. $t_{\text{LOW:SEXT}}$ is the cumulative time a given slave device is allowed to extend the clock cycles in one message from the initial START to the STOP. It is possible that another slave device or the master also extends the clock causing the combined clock low extend time to be greater than $t_{\text{LOW:SEXT}}$. Therefore, this parameter is measured with the slave device as the sole target of a full-speed master.

2. $t_{\text{LOW:MEXT}}$ is the cumulative time a master device is allowed to extend its clock cycles within each byte of a message as defined from START-to-ACK, ACK-to-ACK, or ACK-to-STOP. It is possible that a slave device or another master also extends the clock, causing the combined clock low time to be greater than $t_{\text{LOW:MEXT}}$ on a given byte. Therefore, this parameter is measured with a full speed slave device as the sole target of the master.

**Figure 223. Timeout intervals for $t_{\text{LOW:SEXT}}$, $t_{\text{LOW:MEXT}}$**

![Timeout intervals for $t_{\text{LOW:SEXT}}$, $t_{\text{LOW:MEXT}}$](image-url)
Bus idle detection

A master can assume that the bus is free if it detects that the clock and data signals have been high for $t_{\text{IDLE}} > t_{\text{HIGH,MAX}}$ (refer to I2C timings).

This timing parameter covers the condition where a master has been dynamically added to the bus, and may not have detected a state transition on the SMBCLK or SMBDAT lines. In this case, the master must wait long enough to ensure that a transfer is not currently in progress. The peripheral supports a hardware bus idle detection.

22.4.13 SMBus initialization

This section is relevant only when SMBus feature is supported (see Section 22.3).

In addition to I2C initialization, some other specific initialization must be done to perform SMBus communication.

Received command and data acknowledge control (slave mode)

A SMBus receiver must be able to NACK each received command or data. To allow ACK control in slave mode, the Slave byte control mode must be enabled by setting the SBC bit in the I2C_CR1 register. Refer to Slave byte control mode for more details.

Specific address (slave mode)

The specific SMBus addresses must be enabled if needed. Refer to Bus idle detection for more details.

- The SMBus device default address (0b1100 001) is enabled by setting the SMBDEN bit in the I2C_CR1 register.
- The SMBus host address (0b0001 000) is enabled by setting the SMBHEN bit in the I2C_CR1 register.
- The alert response address (0b0001100) is enabled by setting the ALERTEN bit in the I2C_CR1 register.

Packet error checking

PEC calculation is enabled by setting the PECEN bit in the I2C_CR1 register. Then the PEC transfer is managed with the help of the hardware byte counter NBYTES[7:0] in the I2C_CR2 register. The PECEN bit must be configured before enabling the I2C.

The PEC transfer is managed with the hardware byte counter, so the SBC bit must be set when interfacing the SMBus in slave mode. The PEC is transferred after NBYTES - 1 data have been transferred when the PECBYTE bit is set and the RELOAD bit is cleared. If RELOAD is set, PECBYTE has no effect.

Caution: Changing the PECEN configuration is not allowed when the I2C is enabled.

Table 79. SMBus with PEC configuration

<table>
<thead>
<tr>
<th>Mode</th>
<th>SBC bit</th>
<th>RELOAD bit</th>
<th>AUTOEND bit</th>
<th>PECBYTE bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Tx/Rx NBYTES + PEC+ STOP</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Master Tx/Rx NBYTES + PEC + ReSTART</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Slave Tx/Rx with PEC</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>
Timeout detection

The timeout detection is enabled by setting the TIMOUTEN and TEXTEN bits in the I2C_TIMEOUTR register. The timers must be programmed in such a way that they detect a timeout before the maximum time given in the SMBus specification.

- **tTIMEOUT** check
  
  To enable the **tTIMEOUT** check, the 12-bit **TIMEOUTA[11:0]** bits must be programmed with the timer reload value, to check the **tTIMEOUT** parameter. The **TIDLE** bit must be configured to **0** to detect the SCL low level timeout.

  Then the timer is enabled by setting the TIMOUTEN in the I2C_TIMEOUTR register.

  If **SCL** is tied low for a time greater than **(TIMEOUTA + 1) x 2048 x tI2CCLK**, the **TIMEOUT** flag is set in the I2C_ISR register.

  Refer to **Table 80**.

Caution: Changing the **TIMEOUTA[11:0]** bits and **TIDLE** bit configuration is not allowed when the **TIMEOUTEN** bit is set.

- **tLOW:SEXT and tLOW:MEXT** check
  
  Depending on if the peripheral is configured as a master or as a slave, the 12-bit **TIMEOUTB** timer must be configured to check **tLOW:SEXT** for a slave, and **tLOW:MEXT** for a master. As the standard specifies only a maximum, the user can choose the same value for both. The timer is then enabled by setting the TEXTEN bit in the I2C_TIMEOUTR register.

  If the SMBus peripheral performs a cumulative SCL stretch for a time greater than **(TIMEOUTB + 1) x 2048 x tI2CCLK**, and in the timeout interval described in **Bus idle detection** section, the **TIMEOUT** flag is set in the I2C_ISR register.

  Refer to **Table 81**.

Caution: Changing the **TIMEOUTB** configuration is not allowed when the **TEXTEN** bit is set.

Bus idle detection

To enable the **tIDLE** check, the 12-bit **TIMEOUTA[11:0]** field must be programmed with the timer reload value, to obtain the **tIDLE** parameter. The **TIDLE** bit must be configured to **’1** to detect both SCL and SDA high level timeout. The timer is then enabled by setting the **TIMOUTEN** bit in the I2C_TIMEOUTR register.

If both the SCL and SDA lines remain high for a time greater than **(TIMEOUTA + 1) x 4 x tI2CCLK**, the **TIMEOUT** flag is set in the I2C_ISR register.

Refer to **Table 82**.

Caution: Changing **TIMEOUTA** and **TIDLE** configuration is not allowed when **TIMEOUTEN** is set.
22.4.14 SMBus: I2C_TIMEOUTR register configuration examples

This section is relevant only when SMBus feature is supported. Refer to Section 22.3.

- Configuring the maximum duration of \( t_{\text{TIMEOUT}} \) to 25 ms:
  
  **Table 80. Examples of TIMEOUTA settings (max \( t_{\text{TIMEOUT}} = 25 \text{ ms} \)**

<table>
<thead>
<tr>
<th>( f_{\text{I2CCLK}} )</th>
<th>TIMEOUTA[11:0] bits</th>
<th>TIDLE bit</th>
<th>TIMEOUTEN bit</th>
<th>( t_{\text{TIMEOUT}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MHz</td>
<td>0x61</td>
<td>0</td>
<td>1</td>
<td>( 98 \times 2048 \times 125 \text{ ns} = 25 \text{ ms} )</td>
</tr>
<tr>
<td>16 MHz</td>
<td>0xC3</td>
<td>0</td>
<td>1</td>
<td>( 196 \times 2048 \times 62.5 \text{ ns} = 25 \text{ ms} )</td>
</tr>
<tr>
<td>48 MHz</td>
<td>0x249</td>
<td>0</td>
<td>1</td>
<td>( 586 \times 2048 \times 20.08 \text{ ns} = 25 \text{ ms} )</td>
</tr>
</tbody>
</table>

- Configuring the maximum duration of \( t_{\text{LOW:SEXT}} \) and \( t_{\text{LOW:MEXT}} \) to 8 ms:
  
  **Table 81. Examples of TIMEOUTB settings**

<table>
<thead>
<tr>
<th>( f_{\text{I2CCLK}} )</th>
<th>TIMEOUTB[11:0] bits</th>
<th>TEXTEN bit</th>
<th>( t_{\text{LOW:EXT}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MHz</td>
<td>0x1F</td>
<td>1</td>
<td>( 32 \times 2048 \times 125 \text{ ns} = 8 \text{ ms} )</td>
</tr>
<tr>
<td>16 MHz</td>
<td>0x3F</td>
<td>1</td>
<td>( 64 \times 2048 \times 62.5 \text{ ns} = 8 \text{ ms} )</td>
</tr>
<tr>
<td>48 MHz</td>
<td>0xBB</td>
<td>1</td>
<td>( 188 \times 2048 \times 20.08 \text{ ns} = 8 \text{ ms} )</td>
</tr>
</tbody>
</table>

- Configuring the maximum duration of \( t_{\text{IDLE}} \) to 50 \( \mu \text{s} \)
  
  **Table 82. Examples of TIMEOUTA settings (max \( t_{\text{IDLE}} = 50 \mu \text{s} \)**

<table>
<thead>
<tr>
<th>( f_{\text{I2CCLK}} )</th>
<th>TIMEOUTA[11:0] bits</th>
<th>TIDLE bit</th>
<th>TIMEOUTEN bit</th>
<th>( t_{\text{IDLE}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MHz</td>
<td>0x63</td>
<td>1</td>
<td>1</td>
<td>( 100 \times 4 \times 125 \text{ ns} = 50 \mu \text{s} )</td>
</tr>
<tr>
<td>16 MHz</td>
<td>0xC7</td>
<td>1</td>
<td>1</td>
<td>( 200 \times 4 \times 62.5 \text{ ns} = 50 \mu \text{s} )</td>
</tr>
<tr>
<td>48 MHz</td>
<td>0x257</td>
<td>1</td>
<td>1</td>
<td>( 600 \times 4 \times 20.08 \text{ ns} = 50 \mu \text{s} )</td>
</tr>
</tbody>
</table>

22.4.15 SMBus slave mode

This section is relevant only when the SMBus feature is supported (refer to Section 22.3).

In addition to I2C slave transfer management (refer to Section 22.4.9), additional software flows are provided to support the SMBus.

**SMBus slave transmitter**

When the IP is used in SMBus, SBC must be programmed to 1 to enable the PEC transmission at the end of the programmed number of data bytes. When the PECBYTE bit is set, the number of bytes programmed in NBYTES[7:0] includes the PEC transmission. In that case the total number of TXIS interrupts is NBYTES - 1, and the content of the I2C_PECR register is automatically transmitted if the master requests an extra byte after the NBYTES - 1 data transfer.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.
Figure 224. Transfer sequence flow for SMBus slave transmitter N bytes + PEC

![Flowchart for SMBus slave transmitter sequence flow]

Figure 225. Transfer bus diagrams for SMBus slave transmitter (SBC = 1)

Example SMBus slave transmitter 2 bytes + PEC,

Legend:
- Transmission
- Reception
- SCL stretch

EV1: ADDR ISR: check ADDCODE, program NBYTES=3, set PECBYTE, set ADDRCF
EV2: TXIS ISR: wr data1
EV3: TXIS ISR: wr data2

MS19867V2
SMBus slave receiver

When the I2C is used in SMBus mode, SBC must be programmed to 1 to allow the PEC checking at the end of the programmed number of data bytes. To allow the ACK control of each byte, the reload mode must be selected (RELOAD = 1). Refer to Slave byte control mode for more details.

To check the PEC byte, the RELOAD bit must be cleared and the PECBYTE bit must be set. In this case, after NBYTES - 1 data have been received, the next received byte is compared with the internal I2C_PECR register content. A NACK is automatically generated if the comparison does not match, and an ACK is automatically generated if the comparison matches, whatever the ACK bit value. Once the PEC byte is received, it is copied into the I2C_RXDR register like any other data, and the RXNE flag is set.

In the case of a PEC mismatch, the PECERR flag is set and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

If no ACK software control is needed, the user can program PECBYTE = 1 and, in the same write operation, program NBYTES with the number of bytes to be received in a continuous flow. After NBYTES - 1 are received, the next received byte is checked as being the PEC.

Caution: The PECBYTE bit has no effect when the RELOAD bit is set.
Figure 226. Transfer sequence flow for SMBus slave receiver N bytes + PEC

1. Slave initialization
2. Read `I2C_ISR.ADDR` = 1?
   - Yes: Read ADDCODE and DIR in `I2C_ISR`
     - I2C_CR2.NBYTES = 1, RELOAD = 1
     - PECBYTE = 1
     - Set `I2C_ICR.ADDRCF`
   - No: SCL stretched
3. Read `I2C_ISR.RXNE` = 1?
   - No: Read `I2C_RXDR.RXDATA`
   - Yes: Program `I2C_CR2.NACK` = 0
4. Program `I2C_CR2.NBYTES` = 1
5. N = N - 1
6. N = 1?
   - Yes: Read `I2C_RXDR.RXDATA`
   - No: Program RELOAD = 0
7. NACK = 0 and NBYTES = 1
8. End
This section is relevant only when the SMBus feature is supported (refer to Section 22.3).

In addition to I2C master transfer management (refer to Section 22.4.10), additional software flows are provided to support the SMBus.

### SMBus master transmitter

When the SMBus master wants to transmit the PEC, the PECBYTE bit must be set and the number of bytes must be programmed in the NBYTES[7:0] field, before setting the START bit. In this case the total number of TXIS interrupts is NBYTES - 1. So if the PECBYTE bit is set when NBYTES = 0x1, the content of the I2C_PECR register is automatically transmitted.

If the SMBus master wants to send a STOP condition after the PEC, automatic end mode must be selected (AUTOEND = 1). In this case, the STOP condition automatically follows the PEC transmission.

When the SMBus master wants to send a RESTART condition after the PEC, software mode must be selected (AUTOEND = 0). In this case, once NBYTES - 1 have been
transmitted, the I2C_PECR register content is transmitted and the TC flag is set after the PEC transmission, stretching the SCL line low. The RESTART condition must be programmed in the TC interrupt subroutine.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

**Figure 228. Bus transfer diagrams for SMBus master transmitter**

---

**Example SMBus master transmitter 2 bytes + PEC, automatic end mode (STOP)**

INIT: program Slave address, program NBYTES = 3, AUTOEND=1, set PECBYTE, set START  
EV1: TXIS ISR: wr data1  
EV2: TXIS ISR: wr data2  

---

**Example SMBus master transmitter 2 bytes + PEC, software end mode (RESTART)**

INIT: program Slave address, program NBYTES = 3, AUTOEND=0, set PECBYTE, set START  
EV1: TXIS ISR: wr data1  
EV2: TXIS ISR: wr data2  
EV3: TC ISR: program Slave address, program NBYTES = N, set START
SMBus master receiver

When the SMBus master wants to receive the PEC followed by a STOP at the end of the transfer, automatic end mode can be selected (AUTOEND = 1). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES - 1 data have been received, the next received byte is automatically checked versus the I2C_PECR register content. A NACK response is given to the PEC byte, followed by a STOP condition.

When the SMBus master receiver wants to receive the PEC byte followed by a RESTART condition at the end of the transfer, software mode must be selected (AUTOEND = 0). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES - 1 data have been received, the next received byte is automatically checked versus the I2C_PECR register content. The TC flag is set after the PEC byte reception, stretching the SCL line low. The RESTART condition can be programmed in the TC interrupt subroutine.

Caution: The PECBYTE bit has no effect when the RELOAD bit is set.
22.4.16 Error conditions

The following errors are the conditions that can cause a communication fail.

**Bus error (BERR)**

A bus error is detected when a START or a STOP condition is detected and is not located after a multiple of nine SCL clock pulses. A START or a STOP condition is detected when an SDA edge occurs while SCL is high.

The bus error flag is set only if the I2C is involved in the transfer as master or addressed slave (i.e not during the address phase in slave mode).

In case of a misplaced START or RESTART detection in slave mode, the I2C enters address recognition state like for a correct START condition.
When a bus error is detected, the BERR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

**Arbitration lost (ARLO)**

An arbitration loss is detected when a high level is sent on the SDA line, but a low level is sampled on the SCL rising edge.

- In master mode, arbitration loss is detected during the address phase, data phase and data acknowledge phase. In this case, the SDA and SCL lines are released, the START control bit is cleared by hardware and the master switches automatically to slave mode.
- In slave mode, arbitration loss is detected during data phase and data acknowledge phase. In this case, the transfer is stopped, and the SCL and SDA lines are released.

When an arbitration loss is detected, the ARLO flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

**Overrun/underrun error (OVR)**

An overrun or underrun error is detected in slave mode when NOSTRETCH = 1 and:

- In reception when a new byte is received and the RXDR register has not been read yet. The new received byte is lost, and a NACK is automatically sent as a response to the new byte.
- In transmission:
  - When STOPF = 1 and the first data byte must be sent. The content of the I2C_TXDR register is sent if TXE = 0, 0xFF if not.
  - When a new byte must be sent and the I2C_TXDR register has not been written yet, 0xFF is sent.

When an overrun or underrun error is detected, the OVR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

**Packet error checking error (PECERR)**

This section is relevant only when the SMBus feature is supported (refer to Section 22.3).

A PEC error is detected when the received PEC byte does not match with the I2C_PECR register content. A NACK is automatically sent after the wrong PEC reception.

When a PEC error is detected, the PECERR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.
Timeout error (TIMEOUT)

This section is relevant only when the SMBus feature is supported (refer to Section 22.3).

A timeout error occurs for any of these conditions:

- TIDLE = 0 and SCL remained low for the time defined in the TIMEOUTA[11:0] bits: this is used to detect an SMBus timeout.
- TIDLE = 1 and both SDA and SCL remained high for the time defined in the TIMEOUTA [11:0] bits: this is used to detect a bus idle condition.
- Master cumulative clock low extend time reached the time defined in the TIMEOUTB[11:0] bits (SMBus t\text{LOW:}\.\text{MEXT} parameter).
- Slave cumulative clock low extend time reached the time defined in TIMEOUTB[11:0] bits (SMBus t\text{LOW:}\.\text{SEXT} parameter).

When a timeout violation is detected in master mode, a STOP condition is automatically sent.

When a timeout violation is detected in slave mode, SDA and SCL lines are automatically released.

When a timeout error is detected, the TIMEOUT flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Alert (ALERT)

This section is relevant only when the SMBus feature is supported (refer to Section 22.3).

The ALERT flag is set when the I2C interface is configured as a Host (SMBHEN = 1), the alert pin detection is enabled (ALERTEN = 1) and a falling edge is detected on the SMBA pin. An interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

22.4.17 DMA requests

Transmission using DMA

DMA (direct memory access) can be enabled for transmission by setting the TXDMAEN bit in the I2C_CR1 register. Data is loaded from an SRAM area configured using the DMA peripheral (see Section 10: Direct memory access controller (DMA)) to the I2C_TXDR register whenever the TXIS bit is set.

Only the data are transferred with DMA.

- In master mode: the initialization, the slave address, direction, number of bytes and START bit are programmed by software (the transmitted slave address cannot be transferred with DMA). When all data are transferred using DMA, the DMA must be initialized before setting the START bit. The end of transfer is managed with the NBYTES counter. Refer to Master transmitter.
For a code example refer to A.11.8: I2C configured in master mode to transmit with DMA.

- In slave mode:
  - With NOSTRETCH = 0, when all data are transferred using DMA, the DMA must be initialized before the address match event, or in ADDR interrupt subroutine, before clearing ADDR.
  - With NOSTRETCH = 1, the DMA must be initialized before the address match event.
- For instances supporting SMBus: the PEC transfer is managed with NBYTES counter. Refer to SMBus slave transmitter and SMBus master transmitter.

Note: If DMA is used for transmission, the TXIE bit does not need to be enabled.

Reception using DMA

DMA (direct memory access) can be enabled for reception by setting the RXDMAEN bit in the I2C_CR1 register. Data is loaded from the I2C_RXDR register to an SRAM area configured using the DMA peripheral (refer to Section 10: Direct memory access controller (DMA) on page 149) whenever the RXNE bit is set. Only the data (including PEC) are transferred with DMA.

- In master mode, the initialization, the slave address, direction, number of bytes and START bit are programmed by software. When all data are transferred using DMA, the DMA must be initialized before setting the START bit. The end of transfer is managed with the NBYTES counter..
- In slave mode with NOSTRETCH = 0, when all data are transferred using DMA, the DMA must be initialized before the address match event, or in the ADDR interrupt subroutine, before clearing the ADDR flag.
- If SMBus is supported (see Section 22.3) the PEC transfer is managed with the NBYTES counter. Refer to SMBus slave receiver and SMBus master receiver.

Note: If DMA is used for reception, the RXIE bit does not need to be enabled.

For a code example refer to A.11.9: I2C configured in slave mode to receive with DMA.

22.4.18 Debug mode

When the microcontroller enters debug mode (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.

22.5 I2C low-power modes

Table 83. Effect of low-power modes on the I2C

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>No effect; I2C interrupts cause the device to exit the Sleep mode.</td>
</tr>
<tr>
<td>Stop</td>
<td>The contents of I2C registers are kept. The I2C must be disabled before entering Stop mode.</td>
</tr>
<tr>
<td>Standby</td>
<td>The I2C peripheral is powered down and must be reinitialized after exiting Standby.</td>
</tr>
</tbody>
</table>
# 22.6 I2C interrupts

The following table gives the list of I2C interrupt requests.

<table>
<thead>
<tr>
<th>Interrupt acronym</th>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
<th>Interrupt clear method</th>
<th>Exit Sleep mode</th>
<th>Exit Stop modes</th>
<th>Exit Standby modes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I2C_EV</strong></td>
<td>Receive buffer not empty</td>
<td>RXNE</td>
<td>RXIE</td>
<td>Read I2C_RXDR register</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Transmit buffer interrupt status</td>
<td>TXIS</td>
<td>TXIE</td>
<td>Write I2C_TXDR register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stop detection interrupt flag</td>
<td>STOPF</td>
<td>STOPIE</td>
<td>Write STOPCF = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transfer complete reload</td>
<td>TCR</td>
<td>TCIE</td>
<td>Write I2C_CR2 with NBYTES[7:0] ≠ 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transfer complete</td>
<td>TC</td>
<td></td>
<td>Write START = 1 or STOP = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Address matched</td>
<td>ADDR</td>
<td>ADDRIE</td>
<td>Write ADDRCF=1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NACK reception</td>
<td>NACKF</td>
<td>NACKIE</td>
<td>Write NACKCF=1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>I2C_ER</strong></td>
<td>Bus error</td>
<td>BERR</td>
<td>ERRIE</td>
<td>Write BERRCF = 1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Arbitration loss</td>
<td>ARLO</td>
<td>ERRIE</td>
<td>Write ARLOCF = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Overrun/underrun</td>
<td>OVR</td>
<td>ERRIE</td>
<td>Write OVRCF = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>I2C_ER</strong></td>
<td>PEC error</td>
<td>PECERR</td>
<td>ERRIE</td>
<td>Write PECERRCF = 1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Timeout/ tLOW error</td>
<td>TIMEOUT</td>
<td>ERRIE</td>
<td>Write TIMEOUTCF = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMBus alert</td>
<td>ALERT</td>
<td></td>
<td>Write ALERTCF = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 22.7 I2C registers

Refer to Section 1.2 on page 33 for the list of abbreviations used in register descriptions.

The registers are accessed by words (32-bit).

#### 22.7.1 I2C control register 1 (I2C_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access, until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

<table>
<thead>
<tr>
<th>Bit 31:24</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 31</td>
<td>PECEN: PEC enable</td>
</tr>
<tr>
<td></td>
<td>0: PEC calculation disabled</td>
</tr>
<tr>
<td></td>
<td>1: PEC calculation enabled</td>
</tr>
<tr>
<td>Note:</td>
<td>If the SMBus feature is not supported, this bit is reserved and forced by hardware to 0. Refer to Section 22.3.</td>
</tr>
<tr>
<td>Bit 22</td>
<td>ALERTEN: SMBus alert enable</td>
</tr>
<tr>
<td></td>
<td>0: The SMBus alert pin (SMBA) is not supported in host mode (SMBHEN = 1). In device mode (SMBHEN = 0), the SMBA pin is released and the Alert Response Address header is disabled (0001100x followed by NACK).</td>
</tr>
<tr>
<td></td>
<td>1: The SMBus alert pin is supported in host mode (SMBHEN = 1). In device mode (SMBHEN = 0), the SMBA pin is driven low and the Alert Response Address header is enabled (0001100x followed by ACK).</td>
</tr>
<tr>
<td>Note:</td>
<td>When ALERTEN = 0, the SMBA pin can be used as a standard GPIO.</td>
</tr>
<tr>
<td></td>
<td>If the SMBus feature is not supported, this bit is reserved and forced by hardware to 0. Refer to Section 22.3.</td>
</tr>
<tr>
<td>Bit 21</td>
<td>SMBDEN: SMBus device default address enable</td>
</tr>
<tr>
<td></td>
<td>0: Device default address disabled. Address 0b1100001x is NACKed.</td>
</tr>
<tr>
<td></td>
<td>1: Device default address enabled. Address 0b1100001x is ACKed.</td>
</tr>
<tr>
<td>Note:</td>
<td>If the SMBus feature is not supported, this bit is reserved and forced by hardware to 0. Refer to Section 22.3.</td>
</tr>
<tr>
<td>Bit 20</td>
<td>SMBHEN: SMBus host address enable</td>
</tr>
<tr>
<td></td>
<td>0: Host address disabled. Address 0b0001000x is NACKed.</td>
</tr>
<tr>
<td></td>
<td>1: Host address enabled. Address 0b0001000x is ACKed.</td>
</tr>
<tr>
<td>Note:</td>
<td>If the SMBus feature is not supported, this bit is reserved and forced by hardware to 0. Refer to Section 22.3.</td>
</tr>
</tbody>
</table>
Bit 19 **GCEN**: General call enable
   - 0: General call disabled. Address 0b00000000 is NACKed.
   - 1: General call enabled. Address 0b00000000 is ACKed.

Bit 18 Reserved, must be kept at reset value.

Bit 17 **NOSTRETCH**: Clock stretching disable
   - This bit is used to disable clock stretching in slave mode. It must be kept cleared in master mode.
   - 0: Clock stretching enabled
   - 1: Clock stretching disabled
   
   *Note*: This bit can be programmed only when the I2C is disabled (PE = 0).

Bit 16 **SBC**: Slave byte control
   - This bit is used to enable hardware byte control in slave mode.
   - 0: Slave byte control disabled
   - 1: Slave byte control enabled

Bit 15 **RXDMAEN**: DMA reception requests enable
   - 0: DMA mode disabled for reception
   - 1: DMA mode enabled for reception

Bit 14 **TXDMAEN**: DMA transmission requests enable
   - 0: DMA mode disabled for transmission
   - 1: DMA mode enabled for transmission

Bit 13 Reserved, must be kept at reset value.

Bit 12 **ANOFF**: Analog noise filter OFF
   - 0: Analog noise filter enabled
   - 1: Analog noise filter disabled
   
   *Note*: This bit can be programmed only when the I2C is disabled (PE = 0).

Bits 11:8 **DNF[3:0]**: Digital noise filter
   - These bits are used to configure the digital noise filter on SDA and SCL input. The digital filter filters spikes with a length of up to DNF[3:0] \( \times \ t_{I2CCLK} \)
   - 0000: Digital filter disabled
   - 0001: Digital filter enabled and filtering capability up to one \( t_{I2CCLK} \)
   - ... 
   - 1111: digital filter enabled and filtering capability up to fifteen \( t_{I2CCLK} \)
   
   *Note*: If the analog filter is enabled, the digital filter is added to it. This filter can be programmed only when the I2C is disabled (PE = 0).

Bit 7 **ERRIE**: Error interrupts enable
   - 0: Error detection interrupts disabled
   - 1: Error detection interrupts enabled
   
   *Note*: Any of these errors generate an interrupt:
   - Arbitration loss (ARLO)
   - Bus error detection (BERR)
   - Overrun/underrun (OVR)
   - Timeout detection (TIMEOUT)
   - PEC error detection (PECERR)
   - Alert pin event detection (ALERT)
22.7.2 I2C control register 2 (I2C_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACK</td>
<td>STOP</td>
<td>START</td>
<td>HEAD</td>
<td>ADD10</td>
<td>RD</td>
<td>WRN</td>
<td>SADD[9:0]</td>
<td>rs</td>
<td>rs</td>
<td>rs</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:27 Reserved, must be kept at reset value.
Bit 26 **PECBYTE**: Packet error checking byte

This bit is set by software, and cleared by hardware when the PEC is transferred, or when a
STOP condition or an Address matched is received, also when PE = 0.
0: No PEC transfer
1: PEC transmission/reception is requested

*Note:* Writing 0 to this bit has no effect.

This bit has no effect when RELOAD is set.
This bit has no effect in slave mode when SBC = 0.

Bit 25 **AUTOEND**: Automatic end mode (master mode)

This bit is set and cleared by software.
0: software end mode: TC flag is set when NBYTES data are transferred, stretching SCL low.
1: Automatic end mode: a STOP condition is automatically sent when NBYTES data are transferred.

*Note:* This bit has no effect in slave mode or when the RELOAD bit is set.

Bit 24 **RELOAD**: NBYTES reload mode

This bit is set and cleared by software.
0: The transfer is completed after the NBYTES data transfer (STOP or RESTART follows).
1: The transfer is not completed after the NBYTES data transfer (NBYTES is reloaded). TCR
flag is set when NBYTES data are transferred, stretching SCL low.

Bits 23:16 **NBYTES[7:0]**: Number of bytes

The number of bytes to be transmitted/received is programmed there. This field is don’t care
in slave mode with SBC = 0.

*Note:* Changing these bits when the START bit is set is not allowed.

Bit 15 **NACK**: NACK generation (slave mode)

The bit is set by software, cleared by hardware when the NACK is sent, or when a STOP
condition or an Address matched is received, or when PE = 0.
0: an ACK is sent after current received byte.
1: a NACK is sent after current received byte.

*Note:* Writing 0 to this bit has no effect.

This bit is used only in slave mode: in master receiver mode, NACK is automatically
generated after last byte preceding STOP or RESTART condition, whatever the NACK
bit value.

When an overrun occurs in slave receiver NOSTRETCH mode, a NACK is
automatically generated, whatever the NACK bit value.

When hardware PEC checking is enabled (PECBYTE = 1), the PEC acknowledge value
does not depend on the NACK value.

Bit 14 **STOP**: Stop generation (master mode)

The bit is set by software, cleared by hardware when a STOP condition is detected, or when
PE = 0.

In master mode:
0: No Stop generation
1: Stop generation after current byte transfer

*Note:* Writing 0 to this bit has no effect.
Bit 13 **START**: Start generation

This bit is set by software, and cleared by hardware after the Start followed by the address sequence is sent, by an arbitration loss, by a timeout error detection, or when PE = 0. It can also be cleared by software by writing 1 to the ADDRCF bit in the I2C_ICR register.

0: No Start generation
1: Restart/Start generation:

If the I2C is already in master mode with AUTOEND = 0, setting this bit generates a Repeated start condition when RELOAD = 0, after the end of the NBYTES transfer.

Otherwise, setting this bit generates a START condition once the bus is free.

*Note: Writing 0 to this bit has no effect.*

The START bit can be set even if the bus is BUSY or I2C is in slave mode.

This bit has no effect when RELOAD is set.

Bit 12 **HEAD10R**: 10-bit address header only read direction (master receiver mode)

0: The master sends the complete 10-bit slave address read sequence: Start + 2 bytes 10-bit address in write direction + restart + first seven bits of the 10-bit address in read direction.
1: The master sends only the first seven bits of the 10-bit address, followed by read direction.

*Note: Changing this bit when the START bit is set is not allowed.*

Bit 11 **ADD10**: 10-bit addressing mode (master mode)

0: The master operates in 7-bit addressing mode
1: The master operates in 10-bit addressing mode

*Note: Changing this bit when the START bit is set is not allowed.*

Bit 10 **RD_WRN**: Transfer direction (master mode)

0: Master requests a write transfer
1: Master requests a read transfer

*Note: Changing this bit when the START bit is set is not allowed.*

Bits 9:0 **SADD[9:0]**: Slave address (master mode)

In 7-bit addressing mode (ADD10 = 0):
SADD[7:1] must be written with the 7-bit slave address to be sent. Bits SADD[9], SADD[8] and SADD[0] are don’t care.

In 10-bit addressing mode (ADD10 = 1):
SADD[9:0] must be written with the 10-bit slave address to be sent.

*Note: Changing these bits when the START bit is set is not allowed.*

### 22.7.3 I2C own address 1 register (I2C_OAR1)

Address offset: 0x08

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.
22.7.4 I2C own address 2 register (I2C_OAR2)

Address offset: 0x0C
Reset value: 0x0000_0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access, until the previous one is completed. The latency of the second write access can be up to 2x PCLK1 + 6 x I2CCLK.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bits 31:16 Reserved, must be kept at reset value.

Bit 15 OA2EN: Own address 2 enable
0: Own address 2 disabled. The received slave address OA2 is NACKed.
1: Own address 2 enabled. The received slave address OA2 is ACKed.

Bits 14:11 Reserved, must be kept at reset value.

Bit 10 OA1MODE: Own address 1 10-bit mode
0: Own address 1 is a 7-bit address.
1: Own address 1 is a 10-bit address.

Note: This bit can be written only when OA1EN = 0.

Bits 9:0 OA1[9:0]: Interface own slave address
7-bit addressing mode: OA1[7:1] contains the 7-bit own slave address. Bits OA1[9], OA1[8] and OA1[0] are don't care.
10-bit addressing mode: OA1[9:0] contains the 10-bit own slave address.

Note: These bits can be written only when OA1EN = 0.
22.7.5 I2C timing register (I2C_TIMINGR)

Address offset: 0x10
Reset value: 0x0000 0000
Access: no wait states

### Bits 31:28 PRESC[3:0]
Timing prescaler

This field is used to prescale I2CCLK to generate the clock period t\(_{\text{PRESC}}\) used for data setup and hold counters (refer to I2C timings), and for SCL high and low level counters (refer to I2C master initialization).

\[ t_{\text{PRESC}} = (\text{PRESC} + 1) \times t_{\text{I2CCLK}} \]

Bits 7:4 Reserved, must be kept at reset value.

### Bits 3:0 SCLDEL[3:0]: Data setup time

This field is used to generate a delay \( t_{\text{SCLDEL}} \) between SDA edge and SCL rising edge. In master and in slave modes with NOSTRETCH = 0, the SCL line is stretched low during \( t_{\text{SCLDEL}} \).

\[ t_{\text{SCLDEL}} = (\text{SCLDEL} + 1) \times t_{\text{PRESC}} \]

Note: \( t_{\text{SCLDEL}} \) is used to generate \( t_{\text{SU/DAT}} \) timing.
22.7.6 **I2C timeout register (I2C_TIMEOUTR)**

Address offset: 0x14

Reset value: 0x0000 0000

Access: no wait states, except if a write access occurs while a write access is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>TEXTEN: Extended clock timeout enable</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>27</td>
<td>TimeoutB[11:0]: Bus timeout B</td>
</tr>
<tr>
<td>26</td>
<td>This field is used to configure the cumulative clock extension timeout:</td>
</tr>
<tr>
<td>25</td>
<td>In master mode, the master cumulative clock low extend time (tLOW:MEXT) is detected</td>
</tr>
<tr>
<td>24</td>
<td>In slave mode, the slave cumulative clock low extend time (tLOW:SEXT) is detected</td>
</tr>
<tr>
<td>23</td>
<td>tLOW:EXT = (TIMEOUTB + TIDLE = 01) x 2048 x tI2CCLK</td>
</tr>
<tr>
<td>22</td>
<td>Note: These bits can be written only when TEXTEN = 0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>

**Notes:**
- This register must be configured when the I2C is disabled (PE = 0).
- The STM32CubeMX tool calculates and provides the I2C_TIMINGR content in the I2C Configuration window.
22.7.7 I2C interrupt and status register (I2C_ISR)

Address offset: 0x18
Reset value: 0x0000 0001
Access: no wait states

<table>
<thead>
<tr>
<th>Bit 31:24</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 23:17</td>
<td>ADDCODE[6:0]: Address match code (slave mode)</td>
</tr>
<tr>
<td></td>
<td>These bits are updated with the received address when an address match event occurs (ADDR = 1). In the case of a 10-bit address, ADDCODE provides the 10-bit header followed by the two MSBs of the address.</td>
</tr>
<tr>
<td>Bit 16</td>
<td>DIR: Transfer direction (slave mode)</td>
</tr>
<tr>
<td></td>
<td>This flag is updated when an address match event occurs (ADDR = 1).</td>
</tr>
<tr>
<td></td>
<td>0: Write transfer, slave enters receiver mode.</td>
</tr>
<tr>
<td></td>
<td>1: Read transfer, slave enters transmitter mode.</td>
</tr>
<tr>
<td>Bit 15</td>
<td>BUSY: Bus busy</td>
</tr>
<tr>
<td></td>
<td>This flag indicates that a communication is in progress on the bus. It is set by hardware when a START condition is detected, and cleared by hardware when a STOP condition is detected, or when PE = 0.</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>
### Bit 13: ALERT

**SMBus alert**

This flag is set by hardware when SMBHEN = 1 (SMBus host configuration), ALERTEN = 1 and an SMBALERT event (falling edge) is detected on SMBA pin. It is cleared by software by setting the ALERTCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 12: TIMEOUT

**Timeout or t<sub>LOW</sub> detection flag**

This flag is set by hardware when a timeout or extended clock timeout occurred. It is cleared by software by setting the TIMEOUTCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 11: PECERR

**PEC error in reception**

This flag is set by hardware when the received PEC does not match with the PEC register content. A NACK is automatically sent after the wrong PEC reception. It is cleared by software by setting the PECCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 10: OVR

**Overrun/underrun (slave mode)**

This flag is set by hardware in slave mode with NOSTRETCH = 1, when an overrun/underrun error occurs. It is cleared by software by setting the OVRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 9: ARLO

**Arbitration lost**

This flag is set by hardware in case of arbitration loss. It is cleared by software by setting the ARLOCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 8: BERR

**Bus error**

This flag is set by hardware when a misplaced Start or STOP condition is detected whereas the peripheral is involved in the transfer. The flag is not set during the address phase in slave mode. It is cleared by software by setting the BERRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 7: TCR

**Transfer complete reload**

This flag is set by hardware when RELOAD = 1 and NBYTES data have been transferred. It is cleared by software when NBYTES is written to a non-zero value.

*Note: This bit is cleared by hardware when PE = 0.*

This flag is only for master mode, or for slave mode when the SBC bit is set.

### Bit 6: TC

**Transfer complete (master mode)**

This flag is set by hardware when RELOAD = 0, AUTOEND = 0 and NBYTES data have been transferred. It is cleared by software when START bit or STOP bit is set.

*Note: This bit is cleared by hardware when PE = 0.*

### Bit 5: STOPF

**Stop detection flag**

This flag is set by hardware when a STOP condition is detected on the bus and the peripheral is involved in this transfer:
- as a master, provided that the STOP condition is generated by the peripheral.
- as a slave, provided that the peripheral has been addressed previously during this transfer.

It is cleared by software by setting the STOPCF bit.

*Note: This bit is cleared by hardware when PE = 0.*
Bit 4  **NACKF**: Not acknowledge received flag

This flag is set by hardware when a NACK is received after a byte transmission. It is cleared by software by setting the NACKCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 3  **ADDR**: Address matched (slave mode)

This bit is set by hardware as soon as the received slave address matched with one of the enabled slave addresses. It is cleared by software by setting ADDRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 2  **RXNE**: Receive data register not empty (receivers)

This bit is set by hardware when the received data is copied into the I2C_RXDR register, and is ready to be read. It is cleared when I2C_RXDR is read.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 1  **TXIS**: Transmit interrupt status (transmitters)

This bit is set by hardware when the I2C_TXDR register is empty and the data to be transmitted must be written in the I2C_TXDR register. It is cleared when the next data to be sent is written in the I2C_TXDR register.

This bit can be written to 1 by software only when NOSTRETCH = 1, to generate a TXIS event (interrupt if TXIE = 1 or DMA request if TXDMAEN = 1).

*Note: This bit is cleared by hardware when PE = 0.*

Bit 0  **TXE**: Transmit data register empty (transmitters)

This bit is set by hardware when the I2C_TXDR register is empty. It is cleared when the next data to be sent is written in the I2C_TXDR register.

This bit can be written to 1 by software in order to flush the transmit data register I2C_TXDR.

*Note: This bit is set by hardware when PE = 0.*

### 22.7.8 I2C interrupt clear register (I2C_ICR)

Address offset: 0x1C

Reset value: 0x0000 0000

Access: no wait states

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:14  Reserved, must be kept at reset value.

Bit 13  **ALERTCF**: Alert flag clear

Writing 1 to this bit clears the ALERT flag in the I2C_ISR register.

Bit 12  **TIMOUTCF**: Timeout detection flag clear

Writing 1 to this bit clears the TIMEOUT flag in the I2C_ISR register. Refer to Section 22.3.

Bit 11  **PECCF**: PEC error flag clear

Writing 1 to this bit clears the PECERR flag in the I2C_ISR register.
**22.7.9 I2C PEC register (I2C_PECR)**

Address offset: 0x20

Reset value: 0x0000 0000

Access: no wait states

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **PEC[7:0]**: Packet error checking register

This field contains the internal PEC when PECEN=1.

The PEC is cleared by hardware when PE = 0.
### 22.7.10 I2C receive data register (I2C_RXDR)

Address offset: 0x24  
Reset value: 0x0000 0000  
Access: no wait states

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**RXDATA[7:0]**

| Bits 31:8 Reserved, must be kept at reset value. |
| Bits 7:0 **RXDATA[7:0]**: 8-bit receive data |
| Data byte received from the I2C bus |

### 22.7.11 I2C transmit data register (I2C_TXDR)

Address offset: 0x28  
Reset value: 0x0000 0000  
Access: no wait states

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**TXDATA[7:0]**

| Bits 31:8 Reserved, must be kept at reset value. |
| Bits 7:0 **TXDATA[7:0]**: 8-bit transmit data |
| Data byte to be transmitted to the I2C bus |
| *Note: These bits can be written only when TXE = 1.* |
## 22.7.12 I2C register map

The table below provides the I2C register map and the reset values.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Register name</th>
<th>Reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>I2C_CR1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x04</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>I2C_CR2</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x08</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>I2C_OAR1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x0C</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>I2C_OAR2</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x10</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>I2C_TIMINGR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x14</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>I2C_TIMEOUTR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x18</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>I2C_ISR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x1C</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>I2C_ICR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x20</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>I2C_PECR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x24</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>I2C_RXDR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0x28</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>I2C_TXDR</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
Refer to Section 2.2 on page 37 for the register boundary addresses.
23 Universal synchronous/asynchronous receiver transmitter (USART/UART)

23.1 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 programmable baud rate generator.

It supports synchronous one-way communication and Half-duplex Single-wire communication, as well as multiprocessor communications. It also supports Modem operations (CTS/RTS).

High speed data communication is possible by using the DMA (direct memory access) for multibuffer configuration.

23.2 USART main features

- Full-duplex asynchronous communications
- NRZ standard format (mark/space)
- Configurable oversampling method by 16 or 8 to give flexibility between speed and clock tolerance
- A common programmable transmit and receive baud rate of up to 6 Mbit/s when the clock frequency is 48 MHz and oversampling is by 8
- Convenient baud rate programming
- Auto baud rate detection
- Programmable data word length (8 or 9 bits)
- Programmable data order with MSB-first or LSB-first shifting
- Configurable stop bits (1 or 2 stop bits)
- Synchronous mode and clock output for synchronous communications
- Single-wire Half-duplex communications
- Continuous communications using DMA
- Received/transmitted bytes are buffered in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receiver
- Separate signal polarity control for transmission and reception
- Swappable Tx/Rx pin configuration
- Hardware flow control for modem and RS-485 transceiver
• Communication control/error detection flags
• Parity control:
  – Transmits parity bit
  – Checks parity of received data byte
• Fourteen interrupt sources with flags
• Multiprocessor communications
  The USART enters Mute mode if the address does not match.
• Wake-up from Mute mode (by idle line detection or address mark detection)
## 23.3 USART implementation

Table 86. STM32F0x0 USART features\(^{(1)}\)

<table>
<thead>
<tr>
<th>USART modes/features</th>
<th>STM32F030x4, STM32F030x6</th>
<th>STM32F030x8</th>
<th>STM32F070x6</th>
<th>STM32F070xB</th>
<th>STM32F030xC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>USART1</td>
<td>USART2</td>
<td>USART1</td>
<td>USART2</td>
<td>USART1/USART2</td>
</tr>
<tr>
<td>Hardware flow control for modem</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Continuous communication using DMA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multiprocessor communication</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Synchronous mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Smartcard mode</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Single-wire Half-duplex communication</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IrDA SIR ENDEC block</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LIN mode</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dual clock domain and wake-up from Stop mode</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Receiver timeout interrupt</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Modbus communication</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Auto baud rate detection (supported modes)</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>Driver Enable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>USART data length</td>
<td>8 and 9 bits</td>
<td>7(^{(2)}), 8 and 9 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. X = supported.
2. In 7-bit data length mode, Smartcard mode, LIN master mode and Auto baud rate (0x7F and 0x55 frames) detection are not supported.
23.4 USART functional description

Any USART bidirectional communication requires a minimum of two pins: Receive data In (RX) and Transmit data Out (TX):

- **RX**: Receive data Input.
  This 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, this I/O is used to transmit and receive the data.

Serial data are transmitted and received through these pins in normal USART mode. The frames are comprised of:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 1, 2 stop bits indicating that the frame is complete
- The USART interface uses a baud rate generator
- A status register (USART_ISR)
- Receive and transmit data registers (USART_RDR, USART_TDR)
- A baud rate register (USART_BRR)

Refer to *Section 23.7: USART registers on page 620* for the definitions of each bit.

The following pin is required to interface in synchronous mode:

- **CK**: 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. The clock phase and polarity are software programmable.

The following pins are required in RS232 Hardware flow control mode:

- **CTS**: Clear To Send blocks the data transmission at the end of the current transfer (when high)
- **RTS**: Request to send indicates that the USART is ready to receive data (when low).

The following pin is required in RS485 Hardware control mode:

- **DE**: Driver Enable activates the transmission mode of the external transceiver.

*Note:* DE and RTS share the same pin.
23.4.1 USART character description

The word length can be selected as being either 8 or 9 bits by programming the M bit (M0: bit 12) in the USART_CR1 register (see Figure 231).

- 8-bit character length: M0 = 0
- 9-bit character length: M0 = 1

By default, the signal (TX or RX) is in low state during the start bit. It is in high state during the stop bit.

These values can be inverted, separately for each signal, through polarity configuration control.
An **Idle character** is interpreted as an entire frame of “1”s (the number of “1”s includes 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 2 stop bits.

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 231. Word length programming

#### 9-bit word length (M bit is set), 1 Stop bit

<table>
<thead>
<tr>
<th>Bit0</th>
<th>Bit1</th>
<th>Bit2</th>
<th>Bit3</th>
<th>Bit4</th>
<th>Bit5</th>
<th>Bit6</th>
<th>Bit7</th>
<th>Bit8</th>
<th>Stop bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clock

- **Start bit**
- **Stop bit**

**Idle frame**

**Break frame**

**LBCL bit controls last data clock pulse**

#### 8-bit word length (M bit is reset), 1 Stop bit

<table>
<thead>
<tr>
<th>Bit0</th>
<th>Bit1</th>
<th>Bit2</th>
<th>Bit3</th>
<th>Bit4</th>
<th>Bit5</th>
<th>Bit6</th>
<th>Bit7</th>
<th>Stop bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clock

- **Start bit**
- **Stop bit**

**Idle frame**

**Break frame**

**LBCL bit controls last data clock pulse**

---

### 23.4.2 USART transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. The Transmit Enable bit (TE) must be set in order to activate the transmitter function. The data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

**Character transmission**

During an USART transmission, data shifts out least significant bit first (default configuration) on the TX pin. In this mode, the USART_TDR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see Figure 230).
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: 1 and 2 stop bits.

**Note:**

The TE bit must be set before writing the data to be transmitted to the USART_TDR.

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 is lost.

An idle frame is 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 stop bit:** This is the default value of number of stop bits.
- **2 stop bits:** This is supported by normal USART, Single-wire and Modem modes.

An idle frame transmission will include the stop bits.

A break transmission will be 10 low bits (when M0=0) or 11 low bits (when M0=1) followed by 2 stop bits (see Figure 232). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).

![Figure 232. Configurable stop bits](MSv31887V1)
Character transmission procedure
1. Program the M bit in USART_CR1 to define the word length.
2. Select the desired baud rate using the USART_BRR register.
3. Program the number of stop bits in USART_CR2.
4. Enable the USART by writing the UE bit in USART_CR1 register to 1.
5. Select DMA enable (DMAT) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_TDR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_TDR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

For code example, refer to A.15.1: USART transmitter configuration.

Single byte communication
Clearing the TXE bit is always performed by a write to the transmit data register.
The TXE bit is set by hardware and it indicates:
• The data has been moved from the USART_TDR register to the shift register and the data transmission has started.
• The USART_TDR register is empty.
• The next data can be written in the USART_TDR register without overwriting the previous data.

For code example, refer to A.15.2: USART transmit byte.
This flag generates an interrupt if the TXEIE bit is set.
When a transmission is taking place, a write instruction to the USART_TDR register stores the data in the TDR register; next, the data is copied in the shift register at the end of the currently ongoing transmission.
When no transmission is taking place, a write instruction to the USART_TDR register places the data in the shift register, the data transmission starts, and the TXE bit is set.
If a frame is transmitted (after the stop bit) and the TXE bit is set, the TC bit goes high. An interrupt is generated if the TCIE bit is set in the USART_CR1 register.
After writing the last data in the USART_TDR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low-power mode (see Figure 233: TC/TXE behavior when transmitting).
For code example, refer to A.15.3: USART transfer complete.

**Break characters**

Setting the SBKRQ bit transmits a break character. The break frame length depends on the M bit (see Figure 231).

If a ‘1’ is written to the SBKRQ bit, a break character is sent on the TX line after completing the current character transmission. The SBKF bit is set by the write operation and it is reset by hardware when the break character is completed (during the stop bits after the break character). The USART inserts a logic 1 signal (STOP) for the duration of 2 bits at the end of the break frame to guarantee the recognition of the start bit of the next frame.

In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.

**Idle characters**

Setting the TE bit drives the USART to send an idle frame before the first data frame.

### 23.4.3 USART 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**

The start bit detection sequence is the same when oversampling by 16 or by 8.

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: $1 \ 1 \ 1 \ 0 \ X \ 0 \ X \ 0 \ X \ 0 \ X \ 0 \ X \ 0$.
Figure 234. Start bit detection when oversampling by 16 or 8

Note: If the sequence is not complete, the start bit detection aborts and the receiver returns to the idle state (no flag is set), where it waits for a falling edge.

The start bit is confirmed (RXNE flag set, interrupt generated if RXNEIE=1) if the 3 sampled bits are at 0 (first sampling on the 3rd, 5th and 7th bits finds the 3 bits at 0 and second sampling on the 8th, 9th and 10th bits also finds the 3 bits at 0).

The start bit is validated (RXNE flag set, interrupt generated if RXNEIE=1) but the NF noise flag is set if,

a) for both samplings, 2 out of the 3 sampled bits are at 0 (sampling on the 3rd, 5th and 7th bits and sampling on the 8th, 9th and 10th bits)

or

b) for one of the samplings (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), 2 out of the 3 bits are found at 0.

If neither conditions a. or b. are met, the start detection aborts and the receiver returns to the idle state (no flag is set).
Character reception

During an USART reception, data shifts in least significant bit first (default configuration) through the RX pin. In this mode, the USART_RDR register consists of a buffer (RDR) between the internal bus and the receive shift register.

Character reception procedure

1. Program the M bit in USART_CR1 to define the word length.
2. Select the desired baud rate using the baud rate register USART_BRR
3. Program the number of stop bits in USART_CR2.
4. Enable the USART by writing the UE bit in USART_CR1 register to 1.
5. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

For code example, refer to A.15.4: USART receiver configuration.

When a character is received:

- The RXNE bit is set to indicate 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. PE flag can also be set with RXNE.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read of the Receive data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

For code example, refer to A.15.5: USART receive byte.

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 for a received data 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_RDR is performed.
- The shift register are overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or EIE bit is set.
- The ORE bit is reset by setting the ORECF bit in the ICR register.

Note: The ORE bit, when set, indicates that at least 1 datum 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.

Selecting the clock source and the proper oversampling method

The choice of the clock source is done through the Clock Control system (see Section Reset and clock control (RCC)). The clock source must be chosen before enabling the USART (by setting the UE bit).

The choice of the clock source must be done according to two criteria:
- Possible use of the USART in low-power mode
- Communication speed.

The clock source frequency is $f_{CK}$.

The communication speed range (specially the maximum communication speed) is also determined by the clock source.

The receiver implements different user-configurable oversampling techniques for data recovery by discriminating between valid incoming data and noise. This allows a trade-off between the maximum communication speed and noise/clock inaccuracy immunity.

The oversampling method can be selected by programming the OVER8 bit in the USART_CR1 register and can be either 16 or 8 times the baud rate clock (Figure 235 and Figure 236).

Depending on the application:

- Select oversampling by 8 (OVER8=1) to achieve higher speed (up to $f_{CK}/8$). In this case the maximum receiver tolerance to clock deviation is reduced (refer to Section 23.4.5: Tolerance of the USART receiver to clock deviation on page 606)
- Select oversampling by 16 (OVER8=0) to increase the tolerance of the receiver to clock deviations. In this case, the maximum speed is limited to maximum $f_{CK}/16$ where $f_{CK}$ is the clock source frequency.
Programming the ONEBIT bit in the USART_CR3 register selects the method used to evaluate the logic level. There are two options:

- The majority vote of the three samples in the center of the received bit. In this case, when the 3 samples used for the majority vote are not equal, the NF bit is set.
- A single sample in the center of the received bit.

Depending on the application:

- select the three samples’ majority vote method (ONEBIT=0) when operating in a noisy environment and reject the data when a noise is detected (refer to Figure 87) because this indicates that a glitch occurred during the sampling.
- select the single sample method (ONEBIT=1) when the line is noise-free to increase the receiver’s tolerance to clock deviations (see Section 23.4.5: Tolerance of the USART receiver to clock deviation on page 606). In this case the NF bit will never be set.

When noise is detected in a frame:

- The NF bit is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART_RDR 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 is issued if the EIE bit is set in the USART_CR3 register.

The NF bit is reset by setting NFCF bit in ICR register.

**Figure 235. Data sampling when oversampling by 16**
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_RDR 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 is issued if the EIE bit is set in the USART_CR3 register.

The FE bit is reset by writing 1 to the FECF in the USART_ICR register.
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.

- **1 stop bit:** Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
- **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 is set. The second stop bit is not checked for framing error. The RXNE flag is set at the end of the first stop bit.

### 23.4.4 USART 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 USART_BRR register.

**Equation 1: Baud rate for standard USART (SPI mode included) (OVER8 = 0 or 1)**

In case of oversampling by 16, the equation is:

\[
\text{Tx/Rx baud} = \frac{f_{\text{CK}}}{\text{USARTDIV}}
\]

In case of oversampling by 8, the equation is:

\[
\text{Tx/Rx baud} = \frac{2 \times f_{\text{CK}}}{\text{USARTDIV}}
\]

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

- When OVER8 = 0, BRR = USARTDIV.
- When OVER8 = 1

*Note: The baud counters are updated to the new value in the baud registers after a write operation to USART_BRR. Hence the baud rate register value should not be changed during communication.*

In case of oversampling by 16 or 8, USARTDIV must be greater than or equal to 16d.

**How to derive USARTDIV from USART_BRR register values**

**Example 1**

To obtain 9600 baud with \(f_{\text{CK}} = 8 \text{ MHz}\).

- In case of oversampling by 16:
  \[
  \text{USARTDIV} = 8 \, 000 \, 000/9600 \\
  \text{BRR} = \text{USARTDIV} = 0d833 = 0x341
  \]
- In case of oversampling by 8:
  \[
  \text{USARTDIV} = 2 \times 8 \, 000 \, 000/9600 \\
  \text{USARTDIV} = 1666,66 (0d1667 = 0x683) \\
  \text{BRR}[3:0] = 0x3 \gg 1 = 0x1 \\
  \text{BRR} = 0x681
Example 2

To obtain 921.6 kbaud with \( f_{CK} = 48 \) MHz.

- In case of oversampling by 16:
  \[ \text{USARTDIV} = \frac{48 \, 000 \, 000}{921 \, 600} \]
  \[ \text{BRR} = \text{USARTDIV} = 52d = 34h \]

- In case of oversampling by 8:
  \[ \text{USARTDIV} = \frac{2 \times 48 \, 000 \, 000}{921 \, 600} \]
  \[ \text{USARTDIV} = 104 (104d = 68h) \]
  \[ \text{BRR}[3:0] = \text{USARTDIV}[3:0] >> 1 = 8h >> 1 = 4h \]
  \[ \text{BRR} = 0x64 \]

Table 88. Error calculation for programmed baud rates at \( f_{CK} = 48 \) MHz in both cases of oversampling by 16 or by 8

<table>
<thead>
<tr>
<th>Baud rate</th>
<th>Oversampling by 16 (OVER8 = 0)</th>
<th>Oversampling by 8 (OVER8 = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.No</td>
<td>Desired</td>
<td>Actual</td>
</tr>
<tr>
<td>2</td>
<td>2.4 KBps</td>
<td>2.4 KBps</td>
</tr>
<tr>
<td>3</td>
<td>9.6 KBps</td>
<td>9.6 KBps</td>
</tr>
<tr>
<td>4</td>
<td>19.2 KBps</td>
<td>19.2 KBps</td>
</tr>
<tr>
<td>5</td>
<td>38.4 KBps</td>
<td>38.4 KBps</td>
</tr>
<tr>
<td>6</td>
<td>57.6 KBps</td>
<td>57.62 KBps</td>
</tr>
<tr>
<td>7</td>
<td>115.2 KBps</td>
<td>115.11 KBps</td>
</tr>
<tr>
<td>8</td>
<td>230.4 KBps</td>
<td>230.76 KBps</td>
</tr>
<tr>
<td>9</td>
<td>460.8 KBps</td>
<td>461.54 KBps</td>
</tr>
<tr>
<td>10</td>
<td>921.6 KBps</td>
<td>923.07 KBps</td>
</tr>
<tr>
<td>11</td>
<td>2 MBps</td>
<td>2 MBps</td>
</tr>
<tr>
<td>12</td>
<td>3 MBps</td>
<td>3 MBps</td>
</tr>
<tr>
<td>13</td>
<td>4 MBps</td>
<td>N.A</td>
</tr>
<tr>
<td>14</td>
<td>5 MBps</td>
<td>N.A</td>
</tr>
<tr>
<td>15</td>
<td>6 MBps</td>
<td>N.A</td>
</tr>
</tbody>
</table>

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.
23.4.5 Tolerance of the USART receiver to clock deviation

The asynchronous receiver of the USART works correctly only if the total clock system deviation is less than the tolerance of the USART receiver. The causes which contribute to the total deviation are:

- DTRA: Deviation due to the transmitter error (which also includes the deviation of the transmitter’s local oscillator)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver’s local oscillator
- DTCL: Deviation due to the transmission line (generally due to the transceivers which can introduce an asymmetry between the low-to-high transition timing and the high-to-low transition timing)

\[
\text{DTRA + DQUANT + DREC + DTCL < USART receiver’ s tolerance}
\]

The USART receiver can receive data correctly at up to the maximum tolerated deviation specified in Table 89 and Table 90 depending on the following choices:

- 10- or 11-bit character length defined by the M bit in the USART.CR1 register
- Oversampling by 8 or 16 defined by the OVER8 bit in the USART.CR1 register
- Bits BRR[3:0] of USART_BRR register are equal to or different from 0000.
- Use of 1 bit or 3 bits to sample the data, depending on the value of the ONEBIT bit in the USART.CR3 register.

The data specified in Table 89, and Table 90 may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit durations when M = 0 (11-bit durations when M = 1).

<table>
<thead>
<tr>
<th>M bit</th>
<th>OVER8 bit = 0</th>
<th>OVER8 bit = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONEBIT=0</td>
<td>ONEBIT=1</td>
</tr>
<tr>
<td>0</td>
<td>3.75%</td>
<td>4.375%</td>
</tr>
<tr>
<td>1</td>
<td>3.41%</td>
<td>3.97%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M bit</th>
<th>OVER8 bit = 0</th>
<th>OVER8 bit = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONEBIT=0</td>
<td>ONEBIT=1</td>
</tr>
<tr>
<td>0</td>
<td>3.33%</td>
<td>3.88%</td>
</tr>
<tr>
<td>1</td>
<td>3.03%</td>
<td>3.53%</td>
</tr>
</tbody>
</table>

Note:
23.4.6 USART auto baud rate detection

The USART is able to detect and automatically set the USART_BRR register value based on the reception of one character. Automatic baud rate detection is useful under two circumstances:

- The communication speed of the system is not known in advance
- The system is using a relatively low accuracy clock source and this mechanism allows the correct baud rate to be obtained without measuring the clock deviation.

The clock source frequency must be compatible with the expected communication speed (Oversampling by 16 must be selected for a baud rate between \( f_{CK}/65535 \) and \( f_{CK}/16 \)).

Before activating the auto baud rate detection, the auto baud rate detection mode must be chosen. There are various modes based on different character patterns.

They can be chosen through the ABRMOD[1:0] field in the USART_CR2 register. In these auto baud rate modes, the baud rate is measured several times during the synchronization data reception and each measurement is compared to the previous one.

These modes are:

- **Mode 0**: Any character starting with a bit at 1. In this case the USART measures the duration of the Start bit (falling edge to rising edge).
- **Mode 1**: Any character starting with a 10xx bit pattern. In this case, the USART measures the duration of the Start and of the 1st data bit. The measurement is done falling edge to falling edge, ensuring better accuracy in the case of slow signal slopes. In parallel, another check is performed for each intermediate transition of RX line. An error is generated if the transitions on RX are not sufficiently synchronized with the receiver (the receiver being based on the baud rate calculated on bit 0).

Prior to activating auto baud rate detection, the USART_BRR register must be initialized by writing a non-zero baud rate value.

The automatic baud rate detection is activated by setting the ABREN bit in the USART_CR2 register. The USART will then wait for the first character on the RX line. The auto baud rate operation completion is indicated by the setting of the ABRF flag in the USART_ISR register. If the line is noisy, the correct baud rate detection cannot be guaranteed. In this case the BRR value may be corrupted and the ABRE error flag is set. This also happens if the communication speed is not compatible with the automatic baud rate detection range (bit duration not between 16 and 65536 clock periods (oversampling by 16) and not between 8 and 65536 clock periods (oversampling by 8)).

The RXNE interrupt will signal the end of the operation.

At any later time, the auto baud rate detection may be relaunched by resetting the ABRF flag (by writing a 0).

**Note:** If the USART is disabled (UE=0) during an auto baud rate operation, the BRR value may be corrupted.

23.4.7 Multiprocessor communication using USART

In multiprocessor communication, the following bits are to be kept cleared:

- LINEN bit in the USART_CR2 register,
- HDSEL, IREN and SCEN bits in the USART_CR3 register.
It is possible to perform multiprocessor communication with the USART (with several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output connected to the RX inputs of the other USARTs. 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 order to use the mute mode feature, the MME bit must be set in the USART_CR1 register.

In mute mode:
- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART_ISR register is set to 1. RWU can be controlled automatically by hardware or by software, through the MMRQ bit in the USART_RQR register, 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 MMRQ bit is written to 1 and the RWU is automatically set.

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_ISR register. An example of mute mode behavior using Idle line detection is given in [Figure 237](#).

---

**Figure 237. Mute mode using Idle line detection**

![Figure 237](#)

---

608/775  RM0360 Rev 5
Note: If the MMRQ is set while the IDLE character has already elapsed, mute mode will not be entered (RWU is not set).

If the USART is activated while the line is IDLE, the idle state is detected after the duration of one IDLE frame (not only after the reception of one character frame).

4-bit/7-bit address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a ‘1’ otherwise they are considered as data. In an address byte, the address of the targeted receiver is put in the 4 or 7 LSBs. The choice of 7 or 4-bit address detection is done using the ADDM7 bit. This 4-bit/7-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART_CR2 register.

Note: In 7-bit and 9-bit data modes, address detection is done on 6-bit and 8-bit addresses (ADD[5:0] and ADD[7:0]) respectively.

The USART enters mute mode when an address character is received which does not match its programmed address. In this case, the RWU bit is set by hardware. The RXNE flag is not set for this address byte and no interrupt or DMA request is issued when the USART enters mute mode.

The USART also enters mute mode when the MMRQ bit is written to 1. The RWU bit is also automatically set in this case.

The USART 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.

An example of mute mode behavior using address mark detection is given in Figure 238.

**Figure 238. Mute mode using address mark detection**

In this example, the current address of the receiver is 1 (programmed in the USART_CR2 register)

<table>
<thead>
<tr>
<th>RX</th>
<th>IDLE</th>
<th>Addr=0</th>
<th>Data 1</th>
<th>Data 2</th>
<th>IDLE</th>
<th>Addr=1</th>
<th>Data 3</th>
<th>Data 4</th>
<th>Addr=2</th>
<th>Data 5</th>
</tr>
</thead>
</table>

RWU

MMRQ written to 1 (RXNE was cleared)

Matching address

Non-matching address

MSv31155V1
23.4.8 USART 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 91.

### Table 91. Frame formats

<table>
<thead>
<tr>
<th>M bit</th>
<th>PCE bit</th>
<th>USART frame(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

1. Legends: SB: start bit, STB: stop bit, PB: parity bit. In the data register, the PB is always taking the MSB position (8th or 7th, depending on the M bits value).

#### Even parity

The parity bit is calculated to obtain an even number of “1s” inside the frame of the 7 or 8 LSB bits (depending on M bit value) and the parity bit.

As an example, if data=00110101, and 4 bits are set, then the parity bit is 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 M bit value) and the parity bit.

As an example, if data=00110101 and 4 bits set, then the parity bit is 1 if odd parity is selected (PS bit in USART_CR1 = 1).

#### Parity checking in reception

If the parity check fails, the PE flag is set in the USART_ISR register and an interrupt is generated if PEIE is set in the USART_CR1 register. The PE flag is cleared by software writing 1 to the PECF in the USART_ICR register.

#### Parity generation in transmission

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)).

For code example, refer to A.15.6: USART synchronous mode.

23.4.9 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:

- SCEN, HDSEL and IREN bits in the USART_CR3 register.

In this mode, the USART can be used to control bidirectional synchronous serial communications in master mode. The CK pin is the output of the USART transmitter clock.
No clock pulses are sent to the CK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART_CR2 register, clock pulses are, or are not, generated during the last valid data bit (address mark). The CPOL bit in the USART_CR2 register is used to select the clock polarity, and the CPHA bit in the USART_CR2 register is used to select the phase of the external clock (see Figure 239, Figure 240 and Figure 241).

During the Idle state, preamble and send break, the external CK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as CK 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 CK (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 duration).

Note: The CK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled (TE=1) and data is being transmitted (the data register USART_TDR written). This means that it is not possible to receive synchronous data without transmitting data.

The LBCL, CPOL and CPHA bits have to be selected when the USART is disabled (UE=0) to ensure that the clock pulses function correctly.

For code example, refer to A.15.6: USART synchronous mode.

Figure 239. USART example of synchronous transmission
Figure 240. USART data clock timing diagram (M=0)

Figure 241. USART data clock timing diagram (M=1)
23.4.10 USART Single-wire Half-duplex communication

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:

- CLKEN bit 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 where the TX and RX lines are internally connected. The selection between half- and Full-duplex communication is made with a control bit HDSEL in USART_CR3.

As soon as HDSEL is written to 1:

- The TX and RX lines are internally connected
- The RX pin is no longer used
- The TX pin 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 alternate function open-drain with an external pull-up.

Apart from this, the communication protocol is similar to normal USART mode. Any conflicts on the line must be managed by software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continues as soon as data is written in the data register while the TE bit is set.

For code example, refer to A.15.7: USART DMA.

23.4.11 USART continuous communication in DMA mode

The USART is capable of performing continuous communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

Note: Please refer to Section 23.3: USART implementation on page 592 to determine if the DMA mode is supported. If DMA is not supported, use the USART as explained in Section 23.4.2: USART transmitter or Section 23.4.3: USART receiver. To perform continuous communication, the user can clear the TXE/RXNE flags in the USART_ISR register.

For code example, refer to A.15.7: USART DMA.
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 Section 10: Direct memory access controller (DMA) on page 149) to the USART_TDR 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_TDR register address in the DMA control register to configure it as the destination of the transfer. The data is 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 is loaded into the USART_TDR 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. Clear the TC flag in the USART_ISR register by setting the TCCF bit in the USART_ICR register.
7. 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.

In transmission mode, once the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA_ISR register), the TC flag can be monitored to make sure that the USART communication is complete. This is required to avoid corrupting the last transmission before disabling the USART or entering Stop mode. Software must wait until TC=1. The TC flag remains cleared during all data transfers and it is set by hardware at the end of transmission of the last frame.
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_RDR register to a SRAM area configured using the DMA peripheral (refer to Section 10: Direct memory access controller (DMA) on page 149) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure:

1. Write the USART_RDR register address in the DMA control register to configure it as the source of the transfer. The data is 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 is loaded from USART_RDR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred to 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.
Error flagging and interrupt generation in multibuffer communication

In multibuffer communication if any error occurs during the transaction the error flag is asserted after the current byte. An interrupt is generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in single byte reception, there is a separate error flag interrupt enable bit (EIE bit in the USART_CR3 register), which, if set, enables an interrupt after the current byte if any of these errors occur.

23.4.12 RS232 hardware flow control and RS485 driver enable using USART

It is possible to control the serial data flow between 2 devices by using the CTS input and the RTS output. The Figure 245 shows how to connect 2 devices in this mode:
RS232 RTS and CTS flow control can be enabled independently by writing the RTSE and CTSE bits respectively to 1 (in the USART_CR3 register).

**RS232 RTS flow control**

If the RTS flow control is enabled (RTSE=1), then RTS is deasserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full, RTS is asserted, indicating that the transmission is expected to stop at the end of the current frame. *Figure 246* shows an example of communication with RTS flow control enabled.

![Figure 246. RS232 RTS flow control](MSv68794V1)

**RS232 CTS flow control**

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the CTS input before transmitting the next frame. If CTS is deasserted (tied low), then the next data is transmitted (assuming that data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When CTS is asserted 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 CTS 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. *Figure 247* shows an example of communication with CTS flow control enabled.
**RS485 Driver Enable**

The driver enable feature is enabled by setting bit DEM in the USART_CR3 control register. This allows the user to activate the external transceiver control, through the DE (Driver Enable) signal. The assertion time is the time between the activation of the DE signal and the beginning of the START bit. It is programmed using the DEAT [4:0] bit fields in the USART_CR1 control register. The de-assertion time is the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE signal. It is programmed using the DEDT [4:0] bit fields in the USART_CR1 control register. The polarity of the DE signal can be configured using the DEP bit in the USART_CR3 control register.

In USART, the DEAT and DEDT are expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

### 23.5 USART in low-power modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>No effect. USART interrupt causes the device to exit Sleep mode.</td>
</tr>
<tr>
<td>Stop</td>
<td>The USART is not clocked. It is not functional in Stop mode but its configuration is kept upon wake-up.</td>
</tr>
<tr>
<td>Standby</td>
<td>The USART is powered down and must be reinitialized when the device has exited from Standby mode.</td>
</tr>
</tbody>
</table>
23.6 USART interrupts

The USART interrupt events are connected to the same interrupt vector (see Figure 248).

- During transmission: Transmission Complete, Clear to Send, Transmit data Register empty interrupt.
- During reception: Idle Line detection, Overrun error, Receive data register not empty, Parity error, Noise Flag, Framing Error, Character match, etc.

These events generate an interrupt if the corresponding Enable Control Bit is set.

**Table 93. USART interrupt requests**

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable Control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit data register empty</td>
<td>TXE</td>
<td>TXEIE</td>
</tr>
<tr>
<td>CTS interrupt</td>
<td>CTSIF</td>
<td>CTSIE</td>
</tr>
<tr>
<td>Transmission Complete</td>
<td>TC</td>
<td>TCIE</td>
</tr>
<tr>
<td>Receive data register not empty (data ready to be read)</td>
<td>RXNE</td>
<td>RXNEIE</td>
</tr>
<tr>
<td>Overrun error detected</td>
<td>ORE</td>
<td></td>
</tr>
<tr>
<td>Idle line detected</td>
<td>IDLE</td>
<td>IDLEIE</td>
</tr>
<tr>
<td>Parity error</td>
<td>PE</td>
<td>PEIE</td>
</tr>
<tr>
<td>Noise Flag, Overrun error and Framing Error in multibuffer communication.</td>
<td>NF or ORE or FE</td>
<td>EIE</td>
</tr>
<tr>
<td>Character match</td>
<td>CMF</td>
<td>CMIE</td>
</tr>
<tr>
<td>Receiver timeout</td>
<td>RTOF</td>
<td>RTOIE</td>
</tr>
</tbody>
</table>

**Figure 248. USART interrupt mapping diagram**

![USART interrupt mapping diagram](MSv32635V2)
23.7 **USART registers**

Refer to [Section 1.2 on page 33](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32 bits).

23.7.1 **USART control register 1 (USART_CR1)**

Address offset: 0x00

Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bits 31:29</th>
<th>Reserved, must be kept at reset value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 28</td>
<td><strong>M1</strong>: Word length</td>
</tr>
<tr>
<td></td>
<td>This bit, with bit 12 (M0), determines the word length. It is set or cleared by software.</td>
</tr>
<tr>
<td></td>
<td>M[1:0] = 00: 1 Start bit, 8 data bits, n stop bits</td>
</tr>
<tr>
<td></td>
<td>M[1:0] = 01: 1 Start bit, 9 data bits, n stop bits</td>
</tr>
<tr>
<td></td>
<td>M[1:0] = 10: Reserved</td>
</tr>
<tr>
<td></td>
<td>This bit can only be written when the USART is disabled (UE=0).</td>
</tr>
<tr>
<td>Bit 27</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
<tr>
<td>Bit 26</td>
<td><strong>RTOIE</strong>: Receiver timeout interrupt enable</td>
</tr>
<tr>
<td></td>
<td>This bit is set and cleared by software.</td>
</tr>
<tr>
<td></td>
<td>0: Interrupt is inhibited</td>
</tr>
<tr>
<td></td>
<td>1: An USART interrupt is generated when the RTOF bit is set in the USART_ISR register.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. <a href="#">Section 23.3: USART implementation on page 592</a>.</td>
</tr>
<tr>
<td>Bits 25:21</td>
<td><strong>DEAT[4:0]</strong>: Driver Enable assertion time</td>
</tr>
<tr>
<td></td>
<td>This 5-bit value defines the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).</td>
</tr>
<tr>
<td></td>
<td>This bit field can only be written when the USART is disabled (UE=0).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to <a href="#">Section 23.3: USART implementation on page 592</a>.</td>
</tr>
<tr>
<td>Bits 20:16</td>
<td><strong>DEDT[4:0]</strong>: Driver Enable de-assertion time</td>
</tr>
<tr>
<td></td>
<td>This 5-bit value defines the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).</td>
</tr>
<tr>
<td></td>
<td>If the USART_TDR register is written during the DEDT time, the new data is transmitted only when the DEDT and DEAT times have both elapsed.</td>
</tr>
<tr>
<td></td>
<td>This bit field can only be written when the USART is disabled (UE=0).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to <a href="#">Section 23.3: USART implementation on page 592</a>.</td>
</tr>
</tbody>
</table>
Bit 15 **OVER8**: Oversampling mode
0: Oversampling by 16
1: Oversampling by 8
This bit can only be written when the USART is disabled (UE=0).

*Note:*

Bit 14 **CMIE**: Character match interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: A USART interrupt is generated when the CMF bit is set in the USART_ISR register.

Bit 13 **MME**: Mute mode enable
This bit activates the mute mode function of the USART. When set, the USART can switch between the active and mute modes, as defined by the WAKE bit. It is set and cleared by software.
0: Receiver in active mode permanently
1: Receiver can switch between mute mode and active mode.

Bit 12 **M0**: Word length
This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 data bits, n stop bits
1: 1 Start bit, 9 data bits, n stop bits

Bit 11 **WAKE**: Receiver wake-up method
This bit determines the USART wake-up method from Mute mode. It is set or cleared by software.
0: Idle line
1: Address mark
This bit field can only be written when the USART is disabled (UE=0).

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
This bit field can only be written when the USART is disabled (UE=0).

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 is selected after the current byte.
0: Even parity
1: Odd parity
This bit field can only be written when the USART is disabled (UE=0).

Bit 8 **PEIE**: PE interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: A USART interrupt is generated whenever PE=1 in the USART_ISR register

Bit 7 **TXEIE**: interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: A USART interrupt is generated whenever TXE=1 in the USART_ISR register
Bit 6  **TCIE**: Transmission complete interrupt enable
   This bit is set and cleared by software.
   0: Interrupt is inhibited
   1: A USART interrupt is generated whenever TC=1 in the USART_ISR register

Bit 5  **RXNEIE**: RXNE interrupt enable
   This bit is set and cleared by software.
   0: Interrupt is inhibited
   1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_ISR register

Bit 4  **IDLEIE**: IDLE interrupt enable
   This bit is set and cleared by software.
   0: Interrupt is inhibited
   1: A USART interrupt is generated whenever IDLE=1 in the USART_ISR 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
   
   During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble
   (idle line) after the current word. In order to generate an idle character, the TE must not
   be immediately written to 1.

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  Reserved, must be kept at reset value.

Bit 0  **UE**: USART enable
   When this bit is cleared, the USART prescalers and outputs are stopped immediately, and
   current operations are discarded. The configuration of the USART is kept, but all the status
   flags, in the USART_ISR are set to their default values. This bit is set and cleared by
   software.
   0: USART prescaler and outputs disabled, low-power mode
   1: USART enabled

   **Note:** In order to go into low-power mode without generating errors on the line, the TE bit
   must be reset before and the software must wait for the TC bit in the USART_ISR to be
   set before resetting the UE bit.

   The DMA requests are also reset when UE = 0 so the DMA channel must be disabled
   before resetting the UE bit.
### 23.7.2 USART control register 2 (USART_CR2)

Address offset: 0x04
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit 31:28</th>
<th>ADD[7:4]: Address of the USART node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 27:24</td>
<td>ADD[3:0]: Address of the USART node</td>
</tr>
<tr>
<td>Bit 23</td>
<td>RTOEN: Receiver timeout enable</td>
</tr>
<tr>
<td>Bits 22:21</td>
<td>ABRMOD[1:0]: Auto baud rate mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 21</th>
<th>ABRREN: Auto baud rate enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 20</td>
<td>MSBFIRST: Most significant bit first</td>
</tr>
<tr>
<td>Bit 19</td>
<td>DATAINV: Data inversion</td>
</tr>
<tr>
<td>Bit 18</td>
<td>RXINV: Receiver input inversion</td>
</tr>
<tr>
<td>Bit 17</td>
<td>TXINV: Transmitter input inversion</td>
</tr>
<tr>
<td>Bit 16</td>
<td></td>
</tr>
<tr>
<td>Bit 15</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 14</td>
<td>STOP[1:0]: Stop bit</td>
</tr>
<tr>
<td>Bit 13</td>
<td>LBCL: LSB first</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 11</td>
<td>CPHA: Clock phase</td>
</tr>
<tr>
<td>Bit 10</td>
<td>CPOL: Clock polarity</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 3</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Res.</td>
</tr>
<tr>
<td>Bit 0</td>
<td>Res.</td>
</tr>
</tbody>
</table>

**Bits 31:28 ADD[7:4]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.
This is used in multiprocessor communication during Mute mode or Stop mode, for wake-up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1.
It may also be used for character detection during normal reception. Mute mode inactive. In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match.
This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0).

**Bits 27:24 ADD[3:0]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.
This is used in multiprocessor communication during Mute mode or Stop mode, for wake-up with address mark detection.
This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0).

**Bit 23 RTOEN: Receiver timeout enable**

This bit is set and cleared by software.
0: Receiver timeout feature disabled.
1: Receiver timeout feature enabled.
When this feature is enabled, the RTOF flag in the USART_ISR register is set if the RX line is idle (no reception) for the duration programmed in the RTOR (receiver timeout register).

*Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.*

**Bits 22:21 ABRMOD[1:0]: Auto baud rate mode**

These bits are set and cleared by software.
00: Measurement of the start bit is used to detect the baud rate.
01: Falling edge to falling edge measurement. (the received frame must start with a single bit = 1 -> Frame = Start10xxxxxx)
10: 0x7F frame detection.
11: 0x55 frame detection
This bit field can only be written when ABREN = 0 or the USART is disabled (UE=0).

*Note: If DATAINV=1 and/or MSBFIRST=1 the patterns must be the same on the line, for example 0xAA for MSBFIRST)*

*If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.*
Bit 20 **ABREN**: Auto baud rate enable
   This bit is set and cleared by software.
   0: Auto baud rate detection is disabled.
   1: Auto baud rate detection is enabled.

   *Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.*

Bit 19 **MSBFIRST**: Most significant bit first
   This bit is set and cleared by software.
   0: data is transmitted/received with data bit 0 first, following the start bit.
   1: data is transmitted/received with the MSB (bit 7/8/9) first, following the start bit.
   This bit field can only be written when the USART is disabled (UE=0).

Bit 18 **DATAINV**: Binary data inversion
   This bit is set and cleared by software.
   0: Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
   1: Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.
   This bit field can only be written when the USART is disabled (UE=0).

Bit 17 **TXINV**: TX pin active level inversion
   This bit is set and cleared by software.
   0: TX pin signal works using the standard logic levels (VDD =1/idle, Gnd=0/mark)
   1: TX pin signal values are inverted. (VDD =0/mark, Gnd=1/idle).
   This allows the use of an external inverter on the TX line.
   This bit field can only be written when the USART is disabled (UE=0).

Bit 16 **RXINV**: RX pin active level inversion
   This bit is set and cleared by software.
   0: RX pin signal works using the standard logic levels (VDD =1/idle, Gnd=0/mark)
   1: RX pin signal values are inverted. (VDD =0/mark, Gnd=1/idle).
   This allows the use of an external inverter on the RX line.
   This bit field can only be written when the USART is disabled (UE=0).

Bit 15 **SWAP**: Swap TX/RX pins
   This bit is set and cleared by software.
   0: TX/RX pins are used as defined in standard pinout
   1: The TX and RX pins functions are swapped. This allows to work in the case of a cross-wired connection to another USART.
   This bit field can only be written when the USART is disabled (UE=0).

Bit 14 Reserved, must be kept at reset value.

Bits 13:12 **STOP[1:0]**: STOP bits
   These bits are used for programming the stop bits.
   00: 1 stop bit
   01: Reserved
   10: 2 stop bits
   11: Reserved
   This bit field can only be written when the USART is disabled (UE=0).
Bit 11 **CLKEN**: Clock enable
   - This bit allows the user to enable the CK pin.
   - 0: CK pin disabled
   - 1: CK pin enabled
   - This bit can only be written when the USART is disabled (UE=0).

   **Note:** If synchronous mode is not supported, this bit is reserved and must be kept at reset value.
   Please refer to Section 23.3: USART implementation on page 592.

Bit 10 **CPOL**: Clock polarity
   - This bit allows the user to select the polarity of the clock output on the CK pin in synchronous mode.
   - It works in conjunction with the CPHA bit to produce the desired clock/data relationship
   - 0: Steady low value on CK pin outside transmission window
   - 1: Steady high value on CK pin outside transmission window
   - This bit can only be written when the USART is disabled (UE=0).

   **Note:** If synchronous mode is not supported, this bit is reserved and must be kept at reset value.
   Please refer to Section 23.3: USART implementation on page 592.

Bit 9 **CPHA**: Clock phase
   - This bit is used to select the phase of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see Figure 240 and Figure 241)
   - 0: The first clock transition is the first data capture edge
   - 1: The second clock transition is the first data capture edge
   - This bit can only be written when the USART is disabled (UE=0).

   **Note:** If synchronous mode is not supported, this bit is reserved and must be kept at reset value.
   Please refer to Section 23.3: USART implementation on page 592.

Bit 8 **LBCL**: Last bit clock pulse
   - This bit is used to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the CK pin in synchronous mode.
   - 0: The clock pulse of the last data bit is not output to the CK pin
   - 1: The clock pulse of the last data bit is output to the CK pin

   **Caution:** The last bit is the 7th or 8th or 9th data bit transmitted depending on the 7 or 8 or 9 bit format selected by the M bit in the USART_CR1 register.
   - This bit can only be written when the USART is disabled (UE=0).

   **Note:** If synchronous mode is not supported, this bit is reserved and must be kept at reset value.
   Please refer to Section 23.3: USART implementation on page 592.

Bits 7:5 Reserved, must be kept at reset value.

Bit 4 **ADDM7**: 7-bit Address Detection/4-bit Address Detection
   - This bit is for selection between 4-bit address detection or 7-bit address detection.
   - 0: 4-bit address detection
   - 1: 7-bit address detection (in 8-bit data mode)
   - This bit can only be written when the USART is disabled (UE=0)

   **Note:** In 7-bit and 9-bit data modes, the address detection is done on 6-bit and 8-bit address (ADD[5:0] and ADD[7:0]) respectively.

Bits 3:0 Reserved, must be kept at reset value.

**Note:** The 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.
### 23.7.3 USART control register 3 (USART_CR3)

**Address offset:** 0x08  
**Reset value:** 0x0000 0000

| Bit 31:25 | Reserved, must be kept at reset value. |
| Bit 24 | Reserved, must be kept at reset value. |
| Bit 23 | Reserved, must be kept at reset value. |
| Bit 19:17 | Reserved, must be kept at reset value. |
| Bit 16 | Reserved, must be kept at reset value. |
| Bit 15 | **DEP**: Driver enable polarity selection  
0: DE signal is active high.  
1: DE signal is active low.  
This bit can only be written when the USART is disabled (UE=0).  
*Note:* If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.|
| Bit 14 | **DEM**: Driver enable mode  
This bit allows the user to activate the external transceiver control, through the DE signal.  
0: DE function is disabled.  
1: DE function is enabled. The DE signal is output on the RTS pin.  
This bit can only be written when the USART is disabled (UE=0).  
*Note:* If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value.  
Section 23.3: USART implementation on page 592.|
| Bit 13 | **DDRE**: DMA Disable on Reception Error  
0: DMA is not disabled in case of reception error. The corresponding error flag is set but RXNE is kept 0 preventing from overrun. As a consequence, the DMA request is not asserted, so the erroneous data is not transferred (no DMA request), but next correct received data are transferred.  
1: DMA is disabled following a reception error. The corresponding error flag is set, as well as RXNE. The DMA request is masked until the error flag is cleared. This means that the software must first disable the DMA request (DMAR = 0) or clear RXNE before clearing the error flag.  
This bit can only be written when the USART is disabled (UE=0).  
*Note:* The reception errors are: parity error, framing error or noise error. |
Bit 12 **OVRDIS**: Overrun Disable

This bit is used to disable the receive overrun detection.

- 0: Overrun Error Flag, ORE, is set when received data is not read before receiving new data.
- 1: Overrun functionality is disabled. If new data is received while the RXNE flag is still set the ORE flag is not set and the new received data overwrites the previous content of the USART_RDR register.

This bit can only be written when the USART is disabled (UE=0).

*Note*: This control bit allows checking the communication flow without reading the data.

Bit 11 **ONEBIT**: One sample bit method enable

This bit allows the user to select the sample method. When the one sample bit method is selected the noise detection flag (NF) is disabled.

- 0: Three sample bit method
- 1: One sample bit method

This bit can only be written when the USART is disabled (UE=0).

*Note*: **ONEBIT** feature applies only to data bits, It does not apply to Start bit.

Bit 10 **CTSIE**: CTS interrupt enable

- 0: Interrupt is inhibited
- 1: An interrupt is generated whenever CTSIF=1 in the USART_ISR register

*Note*: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.

Bit 9 **CTSE**: CTS enable

- 0: CTS hardware flow control disabled
- 1: CTS mode enabled, data is only transmitted when the CTS input is deasserted (tied to 0).

If the CTS input is asserted while data is being transmitted, then the transmission is completed before stopping. If data is written into the data register while CTS is asserted, the transmission is postponed until CTS is deasserted.

This bit can only be written when the USART is disabled (UE=0)

*Note*: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.

Bit 8 **RTSE**: RTS enable

- 0: RTS hardware flow control disabled
- 1: RTS output 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 RTS output is deasserted (pulled to 0) when data can be received.

This bit can only be written when the USART is disabled (UE=0).

*Note*: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.

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

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

Bits 5:4 Reserved, must be kept at reset value.
23.7.4 USART baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>HDSEL</th>
<th>Half-duplex selection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Selection of Single-wire Half-duplex mode</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Half duplex mode is not selected</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Half duplex mode is selected</td>
</tr>
</tbody>
</table>
| This bit can only be written when the USART is disabled (UE=0).

Bits 2:1 Reserved, must be kept at reset value.

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 flag (FE=1 or ORE=1 or NF=1 in the USART_ISR register).

0: Interrupt is inhibited
1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USART_ISR register.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 5 1 4 1 3 1 2 1 1 1 0 9 8 7 6 5 4 3 2 1 0

BRR[15:0]

| Bits 31:16 | Reserved, must be kept at reset value.

<table>
<thead>
<tr>
<th>Bits 15:4</th>
<th>BRR[15:4]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Bits 3:0</th>
<th>BRR[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>When OVER8 = 0, BRR[3:0] = USARTDIV[3:0].</td>
<td></td>
</tr>
<tr>
<td>When OVER8 = 1:</td>
<td></td>
</tr>
<tr>
<td>BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.</td>
<td></td>
</tr>
<tr>
<td>BRR[3] must be kept cleared.</td>
<td></td>
</tr>
</tbody>
</table>

23.7.5 USART receiver timeout register (USART_RTOR)

Address offset: 0x14
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>Bit 31:16</th>
<th>RTO[23:16]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Bit 15:0</th>
<th>RTO[15:0]</th>
</tr>
</thead>
</table>

628/775
Bits 31:24  Reserved, must be kept at reset value.

Bits 23:0  **RTO[23:0]**: Receiver timeout value

This bit-field gives the Receiver timeout value in terms of number of bit duration. In standard mode, the RTOF flag is set if, after the last received character, no new start bit is detected for more than the RTO value.

**Note:** This value must only be programmed once per received character.

**Note:** **RTOR can be written on the fly. If the new value is lower than or equal to the counter, the RTOF flag is set.**

This register is reserved and forced by hardware to “0x00000000” when the Receiver timeout feature is not supported. Please refer to Section 23.3: USART implementation on page 592.

### 23.7.6 USART request register (USART_RQR)

Address offset: 0x18

Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:4  Reserved, must be kept at reset value.

Bit 3 **RXFRQ**: Receive data flush request

Writing 1 to this bit clears the RXNE flag. This allows to discard the received data without reading it, and avoid an overrun condition.

Bit 2 **MMRQ**: Mute mode request

Writing 1 to this bit puts the USART in mute mode and sets the RWU flag.

Bit 1 **SBKRQ**: Send break request

Writing 1 to this bit sets the SBKF flag and request to send a BREAK on the line, as soon as the transmit machine is available.

**Note:** In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.

Bit 0 **ABRRQ**: Auto baud rate request

Writing 1 to this bit resets the ABRF and ABRE flags in the USART_ISR and request an automatic baud rate measurement on the next received data frame.

**Note:** If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.
23.7.7 **USART interrupt and status register (USART_ISR)**

Address offset: 0x1C  
Reset value: 0x0200 00C0

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ABRF</td>
<td>ABRE</td>
<td>Res</td>
<td>Res</td>
<td>Res</td>
<td>RTOF</td>
<td>CTS</td>
<td>CTSIF</td>
<td>Res</td>
<td>Res</td>
<td>TXE</td>
<td>TC</td>
<td>RXNE</td>
<td>IDLE</td>
<td>ORE</td>
<td>NF</td>
</tr>
</tbody>
</table>

Bits 31:25  Reserved, must be kept at reset value.  
Bits 24:22  Reserved, must be kept at reset value.  
Bit 21  Reserved, must be kept at reset value  
Bit 20  Reserved, must be kept at reset value.  

**Bit 19 RWU: Receiver wake-up from Mute mode**  
This bit indicates if the USART is in mute mode. It is cleared/set by hardware when a wake-up/mute sequence is recognized. The mute mode control sequence (address or IDLE) is selected by the WAKE bit in the USART_CR1 register.  
When wake-up on IDLE mode is selected, this bit can only be set by software, writing 1 to the MMRQ bit in the USART_RQR register.  
0: Receiver in active mode  
1: Receiver in mute mode  

**Bit 18 SBKF: Send break flag**  
This bit indicates that a send break character was requested. It is set by software, by writing 1 to the SBKRQ bit in the USART_RQR register. It is automatically reset by hardware during the stop bit of break transmission.  
0: No break character is transmitted  
1: Break character is transmitted  

**Bit 17 CMF: Character match flag**  
This bit is set by hardware, when the character defined by ADD[7:0] is received. It is cleared by software, writing 1 to the CMCF in the USART_ICR register.  
An interrupt is generated if CMIE=1 in the USART_CR1 register.  
0: No Character match detected  
1: Character Match detected  

**Bit 16 BUSY: Busy flag**  
This bit is set and reset by hardware. It is active when a communication is ongoing on the RX line (successful start bit detected). It is reset at the end of the reception (successful or not).  
0: USART is idle (no reception)  
1: Reception on going
Bit 15 **ABRF**: Auto baud rate flag

This bit is set by hardware when the automatic baud rate has been set (RXNE will also be set, generating an interrupt if RXNEIE = 1) or when the auto baud rate operation was completed without success (ABRE=1) (ABRE, RXNE and FE are also set in this case)

It is cleared by software, in order to request a new auto baud rate detection, by writing 1 to the ABRRQ in the USART_RQR register.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.*

Bit 14 **ABRE**: Auto baud rate error

This bit is set by hardware if the baud rate measurement failed (baud rate out of range or character comparison failed)

It is cleared by software, by writing 1 to the ABRRQ bit in the USART_RQR register.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.*

Bits 13:12 Reserved, must be kept at reset value.

Bit 11 **RTOF**: Receiver timeout

This bit is set by hardware when the timeout value, programmed in the RTOR register has lapsed, without any communication. It is cleared by software, writing 1 to the RTOCF bit in the USART_ICR register.

An interrupt is generated if RTOIE=1 in the USART_CR1 register.

0: Timeout value not reached
1: Timeout value reached without any data reception

*Note: If a time equal to the value programmed in RTOR register separates 2 characters, RTOF is not set. If this time exceeds this value + 2 sample times (2/16 or 2/8, depending on the oversampling method), RTOF flag is set.

The counter counts even if RE = 0 but RTOF is set only when RE = 1. If the timeout has already elapsed when RE is set, then RTOF is set.

If the USART does not support the Receiver timeout feature, this bit is reserved and kept at reset value.*

Bit 10 **CTS**: CTS flag

This bit is set/reset by hardware. It is an inverted copy of the status of the CTS input pin.

0: CTS line set
1: CTS line reset

*Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.*

Bit 9 **CTSIF**: CTS interrupt flag

This bit is set by hardware when the CTS input toggles, if the CTSE bit is set. It is cleared by software, by writing 1 to the CTSCF bit in the USART_ICR register.

An interrupt is generated if CTSIE=1 in the USART_CR3 register.

0: No change occurred on the CTS status line
1: A change occurred on the CTS status line

*Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.*

Bit 8 Reserved, must be kept at reset value.
Bit 7 TXE: Transmit data register empty
This bit is set by hardware when the content of the USART_TDR register has been
transferred into the shift register. It is cleared by a write to the USART_TDR register.
An interrupt is generated if the TXEIE bit = 1 in the USART_CR1 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 if the transmission of a frame containing data is complete and if
TXE is set. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by
software, writing 1 to the TCCF in the USART_ICR register or by a write to the USART_TDR
register.
An interrupt is generated if TCIE=1 in the USART_CR1 register.
0: Transmission is not complete
1: Transmission is complete
Note: If TE bit is reset and no transmission is on going, the TC bit is set immediately.

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_RDR register. It is cleared by a read to the USART_RDR register. The RXNE
flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register.
An interrupt is generated if RXNEIE=1 in the USART_CR1 register.
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
IDLEIE=1 in the USART_CR1 register. It is cleared by software, writing 1 to the IDLECF in
the USART_ICR 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 (i.e. a new idle line
occurs).
If mute mode is enabled (MME=1), IDLE is set if the USART is not mute (RWU=0),
whatever the mute mode selected by the WAKE bit. If RWU=1, IDLE is not set.

Bit 3 ORE: Overrun error
This bit is set by hardware when the data currently being received in the shift register is
ready to be transferred into the RDR register while RXNE=1. It is cleared by a software,
writing 1 to the ORECF, in the USART_ICR register.
An interrupt is generated if RXNEIE=1 in the USART_CR1 register, or EIE = 1 in the
USART_CR3 register.
0: No overrun error
1: Overrun error is detected
Note: When this bit is set, the RDR register content is not lost but the shift register is
overwritten. An interrupt is generated if the ORE flag is set during multibuffer
communication if the EIE bit is set.
This bit is permanently forced to 0 (no overrun detection) when the OVRDIS bit is set in
the USART_CR3 register.
Bit 2 NF: Noise detection flag
This bit is set by hardware when noise is detected on a received frame. It is cleared by software, writing 1 to the NFCF bit in the USART_ICR register.
0: No noise is detected
1: Noise is detected

Note: This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. An interrupt is generated when the NF flag is set during multibuffer communication if the EIE bit is set.

Note: When the line is noise-free, the NF flag can be disabled by programming the ONEBIT bit to 1 to increase the USART tolerance to deviations (Refer to Section 23.4.5: Tolerance of the USART receiver to clock deviation on page 606).

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 software, writing 1 to the FECF bit in the USART_ICR register.

An interrupt is generated if EIE = 1 in the USART_CR3 register.
0: No Framing error is detected
1: Framing error or break character is detected

Bit 0 PE: Parity error
This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by software, writing 1 to the PECF in the USART_ICR register.

An interrupt is generated if PEIE = 1 in the USART_CR1 register.
0: No parity error
1: Parity error

23.7.8 USART interrupt flag clear register (USART_ICR)
Address offset: 0x20
Reset value: 0x0000 0000

| Bits 31:20 | Reserved, must be kept at reset value. |
| Bits 19:18 | Reserved, must be kept at reset value. |
| Bit 17 CMCF: | Character match clear flag |
| Writing 1 to this bit clears the CMF flag in the USART_ISR register. |
| Bits 16:12 Reserved, must be kept at reset value. |
| Bit 11 RTOCF: | Receiver timeout clear flag |
| Writing 1 to this bit clears the RTOF flag in the USART_ISR register. |
| Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592. |
| Bit 10 Reserved, must be kept at reset value. |
Bit 9 **CTSCF**: CTS clear flag
Writing 1 to this bit clears the CTSIF flag in the USART_ISR register.

*Note*: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 23.3: USART implementation on page 592.

Bit 8 Reserved, must be kept at reset value.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TCCF**: Transmission complete clear flag
Writing 1 to this bit clears the TC flag in the USART_ISR register.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **IDLECF**: Idle line detected clear flag
Writing 1 to this bit clears the IDLE flag in the USART_ISR register.

Bit 3 **ORECF**: Overrun error clear flag
Writing 1 to this bit clears the ORE flag in the USART_ISR register.

Bit 2 **NCF**: Noise detected clear flag
Writing 1 to this bit clears the NF flag in the USART_ISR register.

Bit 1 **FECF**: Framing error clear flag
Writing 1 to this bit clears the FE flag in the USART_ISR register.

Bit 0 **PECF**: Parity error clear flag
Writing 1 to this bit clears the PE flag in the USART_ISR register.

### 23.7.9 USART receive data register (USART_RDR)

Address offset: 0x24

Reset value: 0x0000 0000

![RDR Register](image)

Bits 31:9 Reserved, must be kept at reset value.

Bits 8:0 **RDR[8:0]**: Receive data value
Contains the received data character.
The RDR register provides the parallel interface between the input shift register and the internal bus (see Figure 230).
When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.
23.7.10 UART transmit data register (USART_TDR)

Address offset: 0x28
Reset value: 0x0000 0000

| Bits 31:9 | Reserved, must be kept at reset value. |
| Bits 8:0 | TDR[8:0]: Transmit data value |
| Contains the data character to be transmitted. |
| The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 230). |
| 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. |

Note: This register must be written only when TXE=1.

23.7.11 UART register map

The table below gives the UART register map and reset values.

Table 94. UART register map and reset values
Refer to Section 2.2 on page 37 for the register boundary addresses.
24 Serial peripheral interface (SPI)

24.1 Introduction

The SPI interface can be used to communicate with external devices using the SPI protocol. SPI mode is selectable by software. SPI Motorola mode is selected by default after a device reset.

The serial peripheral interface (SPI) protocol supports half-duplex, full-duplex and simplex synchronous, serial communication with external devices. The interface can be configured as 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.

24.2 SPI main features

- Master or slave operation
- Full-duplex synchronous transfers on three lines
- Half-duplex synchronous transfer on two lines (with bidirectional data line)
- Simplex synchronous transfers on two lines (with unidirectional data line)
- 4 to 16-bit data size selection
- Multimaster mode capability
- 8 master mode baud rate prescalers up to \( f_{PCLK}/2 \)
- Slave mode frequency up to \( f_{PCLK}/2 \)
- 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
- SPI Motorola support
- 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 flags with interrupt capability
- CRC Error flag
- Two 32-bit embedded Rx and Tx FIFOs with DMA capability
- Enhanced TI and NSS pulse modes support
24.3 SPI implementation

The following table describes all the SPI instances and their features embedded in the devices.

Table 95. STM32F0x0 SPI implementation

<table>
<thead>
<tr>
<th></th>
<th>SPI Features</th>
<th>STM32F030x4, STM32F030x6 STM32F070x6</th>
<th>STM32F030x8</th>
<th>STM32F070xB STM32F030xC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPI1</td>
<td>SPI1</td>
<td>SPI2</td>
<td>SPI1</td>
</tr>
<tr>
<td>Enhanced NSSP &amp; TI modes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Hardware CRC calculation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data size configurable</td>
<td>from 4 to 16-bit</td>
<td>from 4 to 16-bit</td>
<td>from 4 to 16-bit</td>
<td>from 4 to 16-bit</td>
</tr>
<tr>
<td>Rx/Tx FIFO size</td>
<td>32-bit</td>
<td>32-bit</td>
<td>32-bit</td>
<td>32-bit</td>
</tr>
<tr>
<td>I²S support</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Wake-up capability from Low-power Sleep</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

24.4 SPI functional description

24.4.1 General description

The SPI allows synchronous, serial communication between the MCU and external devices. Application software can manage the communication by polling the status flag or using dedicated SPI interrupt. The main elements of SPI and their interactions are shown in the following block diagram Figure 249.
Four I/O pins are dedicated to SPI communication with external devices.

- **MISO**: Master In / Slave Out data. In the general case, this pin is used to transmit data in slave mode and receive data in master mode.
- **MOSI**: Master Out / Slave In data. In the general case, this pin is used to transmit data in master mode and receive data in slave mode.
- **SCK**: Serial Clock output pin for SPI masters and input pin for SPI slaves.
- **NSS**: Slave select pin. Depending on the SPI and NSS settings, this pin can be used to either:
  - select an individual slave device for communication
  - synchronize the data frame or
  - detect a conflict between multiple masters

See Section 24.4.5: Slave select (NSS) pin management for details.

The SPI bus allows the communication between one master device and one or more slave devices. The bus consists of at least two wires - one for the clock signal and the other for synchronous data transfer. Other signals can be added depending on the data exchange between SPI nodes and their slave select signal management.

### 24.4.2 Communications between one master and one slave

The SPI allows the MCU to communicate using different configurations, depending on the device targeted and the application requirements. These configurations use 2 or 3 wires (with software NSS management) or 3 or 4 wires (with hardware NSS management). Communication is always initiated by the master.
Full-duplex communication

By default, the SPI is configured for full-duplex communication. In this configuration, the shift registers of the master and slave are linked using two unidirectional lines between the MOSI and the MISO pins. During SPI communication, data is shifted synchronously on the SCK clock edges provided by the master. The master transmits the data to be sent to the slave via the MOSI line and receives data from the slave via the MISO line. When the data frame transfer is complete (all the bits are shifted) the information between the master and slave is exchanged.

![Figure 250. Full-duplex single master/ single slave application](image)

1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see Section 24.4.5: Slave select (NSS) pin management.

Half-duplex communication

The SPI can communicate in half-duplex mode by setting the BIDIMODE bit in the SPIx_CR1 register. In this configuration, one single cross connection line is used to link the shift registers of the master and slave together. During this communication, the data is synchronously shifted between the shift registers on the SCK clock edge in the transfer direction selected reciprocally by both master and slave with the BDIOE bit in their SPIx_CR1 registers. In this configuration, the master’s MISO pin and the slave’s MOSI pin are free for other application uses and act as GPIOs.

![Figure 251. Half-duplex single master/ single slave application](image)

1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see Section 24.4.5: Slave select (NSS) pin management.

2. In this configuration, the master’s MISO pin and the slave’s MOSI pin can be used as GPIOs.

3. A critical situation can happen when communication direction is changed not synchronously between two...
nodes working at bidirectional mode and new transmitter accesses the common data line while former transmitter still keeps an opposite value on the line (the value depends on SPI configuration and communication data). Both nodes then fight while providing opposite output levels on the common line temporary till next node changes its direction settings correspondingly, too. It is suggested to insert a serial resistance between MISO and MOSI pins at this mode to protect the outputs and limit the current blowing between them at this situation.

### Simplex communications

The SPI can communicate in simplex mode by setting the SPI in transmit-only or in receive-only using the RXONLY bit in the SPIx_CR1 register. In this configuration, only one line is used for the transfer between the shift registers of the master and slave. The remaining MISO and MOSI pins pair is not used for communication and can be used as standard GPIOs.

- **Transmit-only mode (RXONLY=0):** The configuration settings are the same as for full-duplex. The application has to ignore the information captured on the unused input pin. This pin can be used as a standard GPIO.

- **Receive-only mode (RXONLY=1):** The application can disable the SPI output function by setting the RXONLY bit. In slave configuration, the MISO output is disabled and the pin can be used as a GPIO. The slave continues to receive data from the MOSI pin while its slave select signal is active (see [24.4.5: Slave select (NSS) pin management](#)). Received data events appear depending on the data buffer configuration. In the master configuration, the MOSI output is disabled and the pin can be used as a GPIO. The clock signal is generated continuously as long as the SPI is enabled. The only way to stop the clock is to clear the RXONLY bit or the SPE bit and wait until the incoming pattern from the MISO pin is finished and fills the data buffer structure, depending on its configuration.

**Figure 252. Simplex single master/single slave application (master in transmit-only/ slave in receive-only mode)**

1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 24.4.5: Slave select (NSS) pin management](#).

2. An accidental input information is captured at the input of transmitter Rx shift register. All the events associated with the transmitter receive flow must be ignored in standard transmit only mode (e.g. OVR flag).

3. In this configuration, both the MISO pins can be used as GPIOs.

**Note:** Any simplex communication can be alternatively replaced by a variant of the half-duplex communication with a constant setting of the transaction direction (bidirectional mode is enabled while BDIO bit is not changed).
24.4.3 Standard multislave communication

In a configuration with two or more independent slaves, the master uses GPIO pins to manage the chip select lines for each slave (see Figure 253). The master must select one of the slaves individually by pulling low the GPIO connected to the slave NSS input. When this is done, a standard master and dedicated slave communication is established.

Figure 253. Master and three independent slaves

1. NSS pin is not used on master side at this configuration. It has to be managed internally (SSM=1, SSI=1) to prevent any MODF error.

2. As MISO pins of the slaves are connected together, all slaves must have the GPIO configuration of their MISO pin set as alternate function open-drain (see I/O alternate function input/output section (GPIO)).

24.4.4 Multimaster communication

Unless SPI bus is not designed for a multimaster capability primarily, the user can use built-in feature which detects a potential conflict between two nodes trying to master the bus at the same time. For this detection, NSS pin is used configured at hardware input mode.
The connection of more than two SPI nodes working at this mode is impossible as only one node can apply its output on a common data line at time.

When nodes are non active, both stay at slave mode by default. Once one node wants to overtake control on the bus, it switches itself into master mode and applies active level on the slave select input of the other node via dedicated GPIO pin. After the session is completed, the active slave select signal is released and the node mastering the bus temporary returns back to passive slave mode waiting for next session start.

If potentially both nodes raised their mastering request at the same time a bus conflict event appears (see mode fault MODF event). Then the user can apply some simple arbitration process (e.g. to postpone next attempt by predefined different time-outs applied at both nodes).

24.4.5 Slave select (NSS) pin management

In slave mode, the NSS works as a standard “chip select” input and lets the slave communicate with the master. In master mode, NSS can be used either as output or input. As an input it can prevent multimaster bus collision, and as an output it can drive a slave select signal of a single slave.

Hardware or software slave select management can be set using the SSM bit in the SPIx_CR1 register:

- **Software NSS management (SSM = 1):** in this configuration, slave select information is driven internally by the SSI bit value in register SPIx_CR1. The external NSS pin is free for other application uses.

- **Hardware NSS management (SSM = 0):** in this case, there are two possible configurations. The configuration used depends on the NSS output configuration (SSOE bit in register SPIx_CR1).
  - **NSS output enable (SSM=0, SSOE = 1):** this configuration is only used when the MCU is set as master. The NSS pin is managed by the hardware. The NSS signal is driven low as soon as the SPI is enabled in master mode (SPE=1), and is kept low until the SPI is disabled (SPE =0). A pulse can be generated between

---

1. The NSS pin is configured at hardware input mode at both nodes. Its active level enables the MISO line output control as the passive node is configured as a slave.

---
continuous communications if NSS pulse mode is activated (NSSP=1). The SPI cannot work in multimaster configuration with this NSS setting.

- **NSS output disable (SSM=0, SSOE = 0):** if the microcontroller is acting as the master on the bus, this configuration allows multimaster capability. If the NSS pin is pulled low in this mode, the SPI enters master mode fault state and the device is automatically reconfigured in slave mode. In slave mode, the NSS pin works as a standard “chip select” input and the slave is selected while NSS line is at low level.

**Figure 255. Hardware/software slave select management**

24.4.6 Communication formats

During SPI communication, receive and transmit operations are performed simultaneously. The serial clock (SCK) synchronizes the shifting and sampling of the information on the data lines. The communication format depends on the clock phase, the clock polarity and the data frame format. To be able to communicate together, the master and slaves devices must follow the same communication format.

**Clock phase and polarity controls**

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPIx_CR1 register. The CPOL (clock polarity) bit controls the idle 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 bit is set, the second edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set). Data are latched on each occurrence of this clock transition type. If the CPHA bit is reset, the first edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is set, rising edge if the CPOL bit is reset). Data are latched on each occurrence of this clock transition type.

The combination of CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

*Figure 256*, shows an SPI full-duplex transfer with the four combinations of the CPHA and CPOL bits.

*Note:* Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit. The idle state of SCK must correspond to the polarity selected in the SPIx_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).

*Figure 256. Data clock timing diagram*

1. The order of data bits depends on LSBFIRST bit setting.
Data frame format

The SPI shift register can be set up to shift out MSB-first or LSB-first, depending on the value of the LSBFIRST bit. The data frame size is chosen by using the DS bits. It can be set from 4-bit up to 16-bit length and the setting applies for both transmission and reception. Whatever the selected data frame size, read access to the FIFO must be aligned with the FRXTH level. When the SPIx_DR register is accessed, data frames are always right-aligned into either a byte (if the data fits into a byte) or a half-word (see Figure 257). During communication, only bits within the data frame are clocked and transferred.

Figure 257. Data alignment when data length is not equal to 8-bit or 16-bit

Note: The minimum data length is 4 bits. If a data length of less than 4 bits is selected, it is forced to an 8-bit data frame size.

24.4.7 Configuration of SPI

The configuration procedure is almost the same for master and slave. For specific mode setups, follow the dedicated sections. When a standard communication is to be initialized, perform these steps:

1. Write proper GPIO registers: Configure GPIO for MOSI, MISO and SCK pins.
2. Write to the SPI_CR1 register:
   a) Configure the serial clock baud rate using the BR[2:0] bits (Note: 4).
   b) Configure the CPOL and CPHA bits combination to define one of the four relationships between the data transfer and the serial clock (CPHA must be cleared in NSSP mode). (Note: 2 - except the case when CRC is enabled at TI mode).
   c) Select simplex or half-duplex mode by configuring RXONLY or BIDIMODE and BIDIOE (RXONLY and BIDIMODE cannot be set at the same time).
   d) Configure the LSBFIRST bit to define the frame format (Note: 2).
   e) Configure the CRCL and CRCEN bits if CRC is needed (while SCK clock signal is at idle state).
   f) Configure SSM and SSI (Notes: 2 & 3).
   g) Configure the MSTR bit (in multimaster NSS configuration, avoid conflict state on
3. Write to SPI_CR2 register:
   a) Configure the DS[3:0] bits to select the data length for the transfer.
   b) Configure SSOE (Notes: 1 & 2 & 3).
   c) Set the FRF bit if the TI protocol is required (keep NSSP bit cleared in TI mode).
   d) Set the NSSP bit if the NSS pulse mode between two data units is required (keep CHPA and TI bits cleared in NSSP mode).
   e) Configure the FRXTH bit. The RXFIFO threshold must be aligned to the read access size for the SPIx_DR register.
   f) Initialize LDMA_TX and LDMA_RX bits if DMA is used in packed mode.

4. Write to SPI_CRCPR register: Configure the CRC polynomial if needed.

5. Write proper DMA registers: Configure DMA streams dedicated for SPI Tx and Rx in DMA registers if the DMA streams are used.

Note: (1) Step is not required in slave mode.
      (2) Step is not required in TI mode.
      (3) Step is not required in NSSP mode.
      (4) The step is not required in slave mode except slave working at TI mode

For code example refer to the Appendix sections A.14.1: SPI master configuration and A.14.2: SPI slave configuration.

24.4.8 Procedure for enabling SPI

It is recommended to enable the SPI slave before the master sends the clock. If not, undesired data transmission might occur. The data register of the slave must already contain data to be sent before starting communication with the master (either on the first edge of the communication clock, or before the end of the ongoing communication if the clock signal is continuous). The SCK signal must be settled at an idle state level corresponding to the selected polarity before the SPI slave is enabled.

The master at full-duplex (or in any transmit-only mode) starts to communicate when the SPI is enabled and TXFIFO is not empty, or with the next write to TXFIFO.

In any master receive only mode (RXONLY = 1 or BIDIMODE = 1 & BIDIOE = 0), master starts to communicate and the clock starts running immediately after SPI is enabled.

For handling DMA, follow the dedicated section.

24.4.9 Data transmission and reception procedures

RXFIFO and TXFIFO

All SPI data transactions pass through the 32-bit embedded FIFOs. This enables the SPI to work in a continuous flow, and prevents overruns when the data frame size is short. Each direction has its own FIFO called TXFIFO and RXFIFO. These FIFOs are used in all SPI modes except for receiver-only mode (slave or master) with CRC calculation enabled (see Section 24.4.14: CRC calculation).

The handling of FIFOs depends on the data exchange mode (duplex, simplex), data frame format (number of bits in the frame), access size performed on the FIFO data registers (8-bit
or 16-bit), and whether or not data packing is used when accessing the FIFOs (see Section 24.4.13: TI mode).

A read access to the SPIx_DR register returns the oldest value stored in RXFIFO that has not been read yet. A write access to the SPIx_DR stores the written data in the TXFIFO at the end of a send queue. The read access must be always aligned with the RXFIFO threshold configured by the FRXTH bit in SPIx_CR2 register. FTLVL[1:0] and FRLVL[1:0] bits indicate the current occupancy level of both FIFOs.

A read access to the SPIx DR register must be managed by the RXNE event. This event is triggered when data is stored in RXFIFO and the threshold (defined by FRXTH bit) is reached. When RXNE is cleared, RXFIFO is considered to be empty. In a similar way, write access of a data frame to be transmitted is managed by the TXE event. This event is triggered when the TXFIFO level is less than or equal to half of its capacity. Otherwise TXE is cleared and the TXFIFO is considered as full. In this way, RXFIFO can store up to four data frames, whereas TXFIFO can only store up to three when the data frame format is not greater than 8 bits. This difference prevents possible corruption of 3x 8-bit data frames already stored in the TXFIFO when software tries to write more data in 16-bit mode into TXFIFO. Both TXE and RXNE events can be polled or handled by interrupts. See Figure 259 through Figure 262.

Another way to manage the data exchange is to use DMA (see Communication using DMA (direct memory addressing)).

If the next data is received when the RXFIFO is full, an overrun event occurs (see description of OVR flag at Section 24.4.10: SPI status flags). An overrun event can be polled or handled by an interrupt.

The BSY bit being set indicates ongoing transaction of a current data frame. When the clock signal runs continuously, the BSY flag stays set between data frames at master but becomes low for a minimum duration of one SPI clock at slave between each data frame transfer.

**Sequence handling**

A few data frames can be passed at single sequence to complete a message. When transmission is enabled, a sequence begins and continues while any data is present in the TXFIFO of the master. The clock signal is provided continuously by the master until TXFIFO becomes empty, then it stops waiting for additional data.

In receive-only modes, half-duplex (BIDIMODE=1, BIDIOE=0) or simplex (BIDIMODE=0, RXONLY=1) the master starts the sequence immediately when both SPI is enabled and receive-only mode is activated. The clock signal is provided by the master and it does not stop until either SPI or receive-only mode is disabled by the master. The master receives data frames continuously up to this moment.

While the master can provide all the transactions in continuous mode (SCK signal is continuous) it has to respect slave capability to handle data flow and its content at anytime. When necessary, the master must slow down the communication and provide either a slower clock or separate frames or data sessions with sufficient delays. Be aware there is no underflow error signal for master or slave in SPI mode, and data from the slave is always transacted and processed by the master even if the slave could not prepare it correctly in time. It is preferable for the slave to use DMA, especially when data frames are shorter and bus rate is high.

Each sequence must be encased by the NSS pulse in parallel with the multislave system to select just one of the slaves for communication. In a single slave system it is not necessary
to control the slave with NSS, but it is often better to provide the pulse here too, to synchronize the slave with the beginning of each data sequence. NSS can be managed by both software and hardware (see Section 24.4.5: Slave select (NSS) pin management).

When the BSY bit is set it signifies an ongoing data frame transaction. When the dedicated frame transaction is finished, the RXNE flag is raised. The last bit is just sampled and the complete data frame is stored in the RXFIFO.

Procedure for disabling the SPI

When SPI is disabled, it is mandatory to follow the disable procedures described in this paragraph. It is important to do this before the system enters a low-power mode when the peripheral clock is stopped. Ongoing transactions can be corrupted in this case. In some modes the disable procedure is the only way to stop continuous communication running.

Master in full-duplex or transmit only mode can finish any transaction when it stops providing data for transmission. In this case, the clock stops after the last data transaction. Special care must be taken in packing mode when an odd number of data frames are transacted to prevent some dummy byte exchange (refer to Data packing section). Before the SPI is disabled in these modes, the user must follow standard disable procedure. When the SPI is disabled at the master transmitter while a frame transaction is ongoing or next data frame is stored in TXFIFO, the SPI behavior is not guaranteed.

When the master is in any receive only mode, the only way to stop the continuous clock is to disable the peripheral by SPE=0. This must occur in specific time window within last data frame transaction just between the sampling time of its first bit and before its last bit transfer starts (in order to receive a complete number of expected data frames and to prevent any additional “dummy” data reading after the last valid data frame). Specific procedure must be followed when disabling SPI in this mode.

Data received but not read remains stored in RXFIFO when the SPI is disabled, and must be processed the next time the SPI is enabled, before starting a new sequence. To prevent having unread data, ensure that RXFIFO is empty when disabling the SPI, by using the correct disabling procedure, or by initializing all the SPI registers with a software reset via the control of a specific register dedicated to peripheral reset (see the SPIiRST bits in the RCC_APB1RSTR registers).

Standard disable procedure is based on pulling BSY status together with FTLVL[1:0] to check if a transmission session is fully completed. This check can be done in specific cases, too, when it is necessary to identify the end of ongoing transactions, for example:

- When NSS signal is managed by software and master has to provide proper end of NSS pulse for slave, or
- When transactions’ streams from DMA or FIFO are completed while the last data frame or CRC frame transaction is still ongoing in the peripheral bus.

The correct disable procedure is (except when receive only mode is used):
1. Wait until FTLVL[1:0] = 00 (no more data to transmit).
2. Wait until BSY=0 (the last data frame is processed).
3. Disable the SPI (SPE=0).
4. Read data until FRLVL[1:0] = 00 (read all the received data).

The correct disable procedure for certain receive only modes is:
1. Interrupt the receive flow by disabling SPI (SPE=0) in the specific time window while the last data frame is ongoing.
2. Wait until BSY=0 (the last data frame is processed).
3. Read data until FRLVL[1:0] = 00 (read all the received data).

Note: If packing mode is used and an odd number of data frames with a format less than or equal to 8 bits (fitting into one byte) has to be received, FRXTH must be set when FRLVL[1:0] = 01, in order to generate the RXNE event to read the last odd data frame and to keep good FIFO pointer alignment.

**Data packing**

When the data frame size fits into one byte (less than or equal to 8 bits), data packing is used automatically when any read or write 16-bit access is performed on the SPIx_DR register. The double data frame pattern is handled in parallel in this case. At first, the SPI operates using the pattern stored in the LSB of the accessed word, then with the other half stored in the MSB. *Figure 258* provides an example of data packing mode sequence handling. Two data frames are sent after the single 16-bit access the SPIx_DR register of the transmitter. This sequence can generate just one RXNE event in the receiver if the RXFIFO threshold is set to 16 bits (FRXTH=0). The receiver then has to access both data frames by a single 16-bit read of SPIx_DR as a response to this single RXNE event. The RxFIFO threshold setting and the following read access must be always kept aligned at the receiver side, as data can be lost if it is not in line.

A specific problem appears if an odd number of such “fit into one byte” data frames must be handled. On the transmitter side, writing the last data frame of any odd sequence with an 8-bit access to SPIx_DR is enough. The receiver has to change the Rx_FIFO threshold level for the last data frame received in the odd sequence of frames in order to generate the RXNE event.

*Figure 258. Packing data in FIFO for transmission and reception*

**Communication using DMA (direct memory addressing)**

To operate at its maximum speed and to facilitate the data register read/write process required to avoid overrun, the SPI features a DMA capability, which implements a simple request/acknowledge protocol.
A DMA access is requested when the TXDMAEN or RXDMAEN enable bit in the SPIx_CR2 register is set. Separate requests must be issued to the Tx and Rx buffers.

- In transmission, a DMA request is issued each time TXE is set to 1. The DMA then writes to the SPIx_DR register.
- In reception, a DMA request is issued each time RXNE is set to 1. The DMA then reads the SPIx_DR register.

See Figure 259 through Figure 262.

When the SPI is used only to transmit data, it is possible to enable only the SPI Tx DMA channel. In this case, the OVR flag is set because the data received is not read. When the SPI is used only to receive data, it is possible to enable only the SPI Rx DMA channel.

In transmission mode, when the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA_ISR register), the BSY flag can be monitored to ensure that the SPI communication is complete. This is required to avoid corrupting the last transmission before disabling the SPI or entering the Stop mode. The software must first wait until FTLVL[1:0]=00 and then until BSY=0.

When starting communication using DMA, to prevent DMA channel management raising error events, these steps must be followed in order:

1. Enable DMA Rx buffer in the RXDMAEN bit in the SPI_CR2 register, if DMA Rx is used.
2. Enable DMA streams for Tx and Rx in DMA registers, if the streams are used.
3. Enable DMA Tx buffer in the TXDMAEN bit in the SPI_CR2 register, if DMA Tx is used.
4. Enable the SPI by setting the SPE bit.

For code example refer to the Appendix sections A.14.5: SPI master configuration with DMA and A.14.6: SPI slave configuration with DMA.

To close communication it is mandatory to follow these steps in order:

1. Disable DMA streams for Tx and Rx in the DMA registers, if the streams are used.
2. Disable the SPI by following the SPI disable procedure.
3. Disable DMA Tx and Rx buffers by clearing the TXDMAEN and RXDMAEN bits in the SPI_CR2 register, if DMA Tx and/or DMA Rx are used.

### Packing with DMA

If the transfers are managed by DMA (TXDMAEN and RXDMAEN set in the SPIx_CR2 register) packing mode is enabled/disabled automatically depending on the PSIZE value configured for SPI TX and the SPI RX DMA channel. If the DMA channel PSIZE value is equal to 16-bit and SPI data size is less than or equal to 8-bit, then packing mode is enabled. The DMA then automatically manages the write operations to the SPIx_DR register.

If data packing mode is used and the number of data to transfer is not a multiple of two, the LDMA_TX/LDMA_RX bits must be set. The SPI then considers only one data for the transmission or reception to serve the last DMA transfer (for more details refer to Data packing on page 650.)
Communication diagrams

Some typical timing schemes are explained in this section. These schemes are valid no matter if the SPI events are handled by polling, interrupts or DMA. For simplicity, the LSBFIRST=0, CPOL=0 and CPHA=1 setting is used as a common assumption here. No complete configuration of DMA streams is provided.

The following numbered notes are common for Figure 259 on page 653 through Figure 262 on page 656:

1. The slave starts to control MISO line as NSS is active and SPI is enabled, and is disconnected from the line when one of them is released. Sufficient time must be provided for the slave to prepare data dedicated to the master in advance before its transaction starts.
   At the master, the SPI peripheral takes control at MOSI and SCK signals (occasionally at NSS signal as well) only if SPI is enabled. If SPI is disabled the SPI peripheral is disconnected from GPIO logic, so the levels at these lines depends on GPIO setting exclusively.

2. At the master, BSY stays active between frames if the communication (clock signal) is continuous. At the slave, BSY signal always goes down for at least one clock cycle between data frames.

3. The TXE signal is cleared only if TXFIFO is full.

4. The DMA arbitration process starts just after the TXDMAEN bit is set. The TXE interrupt is generated just after the TXEIE is set. As the TXE signal is at an active level, data transfers to TxFIFO start, until TxFIFO becomes full or the DMA transfer completes.

5. If all the data to be sent can fit into TxFIFO, the DMA Tx TCIF flag can be raised even before communication on the SPI bus starts. This flag always rises before the SPI transaction is completed.

6. The CRC value for a package is calculated continuously frame by frame in the SPIx_TXCRCR and SPIx_RXCRCR registers. The CRC information is processed after the entire data package has completed, either automatically by DMA (Tx channel must be set to the number of data frames to be processed) or by SW (the user must handle CRCNEXT bit during the last data frame processing).
   While the CRC value calculated in SPIx_TXCRCR is simply sent out by transmitter, received CRC information is loaded into RxFIFO and then compared with the SPIx_RXCRCR register content (CRC error flag can be raised here if any difference).
   This is why the user must take care to flush this information from the FIFO, either by software reading out all the stored content of RxFIFO, or by DMA when the proper number of data frames is preset for Rx channel (number of data frames + number of CRC frames) (see the settings at the example assumption).

7. In data packed mode, TxE and RxNE events are paired and each read/write access to the FIFO is 16 bits wide until the number of data frames are even. If the TxFIFO is ¾ full FTLVL status stays at FIFO full level. That is why the last odd data frame cannot be stored before the TxFIFO becomes ½ full. This frame is stored into TxFIFO with an 8-bit access either by software or automatically by DMA when LDMA_TX control is set.

8. To receive the last odd data frame in packed mode, the Rx threshold must be changed to 8-bit when the last data frame is processed, either by software setting FRXTH=1 or automatically by a DMA internal signal when LDMA_RX is set.
Assumptions for master full-duplex communication example:

- Data size > 8 bit

If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also: Communication diagrams on page 652 for details about common assumptions and notes.
Assumptions for slave full-duplex communication example:

- Data size > 8 bit

If DMA is used:
- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also Communication diagrams on page 652 for details about common assumptions and notes.

For code example refer to the Appendix section A.14.3: SPI full duplex communication.
Assumptions for master full-duplex communication with CRC example:

- Data size = 16 bit
- CRC enabled

If DMA is used:
- Number of Tx frames transacted by DMA is set to 2
- Number of Rx frames transacted by DMA is set to 3

See also: Communication diagrams on page 652 for details about common assumptions and notes.
Figure 262. Master full-duplex communication in packed mode

Assumptions for master full-duplex communication in packed mode example:

- Data size = 5 bit
- Read/write FIFO is performed mostly by 16-bit access
- FRXTH=0

If DMA is used:

- Number of Tx frames to be transacted by DMA is set to 3
- Number of Rx frames to be transacted by DMA is set to 3
- PSIZE for both Tx and Rx DMA channel is set to 16-bit
- LDMA_TX=1 and LDMA_RX=1

See also: Communication diagrams on page 652 for details about common assumptions and notes.
24.4.10 SPI status flags

Three status flags are provided for the application to completely monitor the state of the SPI bus.

Tx buffer empty flag (TXE)

The TXE flag is set when transmission TXFIFO has enough space to store data to send. TXE flag is linked to the TXFIFO level. The flag goes high and stays high until the TXFIFO level is lower or equal to 1/2 of the FIFO depth. An interrupt can be generated if the TXEIE bit in the SPIx_CR2 register is set. The bit is cleared automatically when the TXFIFO level becomes greater than 1/2.

Rx buffer not empty (RXNE)

The RXNE flag is set depending on the FRXTH bit value in the SPIx_CR2 register:

- If FRXTH is set, RXNE goes high and stays high until the RXFIFO level is greater or equal to 1/4 (8-bit).
- If FRXTH is cleared, RXNE goes high and stays high until the RXFIFO level is greater than or equal to 1/2 (16-bit).

An interrupt can be generated if the RXNEIE bit in the SPIx_CR2 register is set.

The RXNE is cleared by hardware automatically when the above conditions are no longer true.

Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect).

When BSY is set, it indicates that a data transfer is in progress on the SPI (the SPI bus is busy).

The BSY flag can be used in certain modes to detect the end of a transfer so that the software can disable the SPI or its peripheral clock before entering a low-power mode which does not provide a clock for the peripheral. This avoids corrupting the last transfer.

The BSY flag is also useful for preventing write collisions in a multimaster system.

The BSY flag is cleared under any one of the following conditions:

- When the SPI is correctly disabled
- When a fault is detected in Master mode (MODF bit set to 1)
- In Master mode, when it finishes a data transmission and no new data is ready to be sent
- In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.

Note: When the next transmission can be handled immediately by the master (e.g. if the master is in Receive-only mode or its Transmit FIFO is not empty), communication is continuous and the BSY flag remains set to '1' between transfers on the master side. Although this is not the case with a slave, it is recommended to use always the TXE and RXNE flags (instead of the BSY flags) to handle data transmission or reception operations.
24.4.11 SPI error flags

An SPI interrupt is generated if one of the following error flags is set and interrupt is enabled by setting the ERRIE bit.

Overrun flag (OVR)

An overrun condition occurs when data is received by a master or slave and the RXFIFO has not enough space to store this received data. This can happen if the software or the DMA did not have enough time to read the previously received data (stored in the RXFIFO) or when space for data storage is limited e.g. the RXFIFO is not available when CRC is enabled in receive only mode so in this case the reception buffer is limited into a single data frame buffer (see Section 24.4.14: CRC calculation).

When an overrun condition occurs, the newly received value does not overwrite the previous one in the RXFIFO. The newly received value is discarded and all data transmitted subsequently is lost. Clearing the OVR bit is done by a read access to the SPI_DR register followed by a read access to the SPI_SR register.

Mode fault (MODF)

Mode fault occurs when the master device has its internal NSS signal (NSS pin in NSS hardware mode, or SSI bit in NSS software mode) pulled low. This automatically sets the MODF bit. Master mode fault affects the SPI interface 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 cleared. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is cleared, 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 SPIx_SR register while the MODF bit is set.
2. Then write to the SPIx_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 after this clearing sequence. As a security, hardware does not allow the SPE and MSTR bits to be set while the MODF bit is set. In a slave device the MODF bit cannot be set except as the result of a previous multimaster conflict.

CRC error (CRCERR)

This flag is used to verify the validity of the value received when the CRCEN bit in the SPIx_CR1 register is set. The CRCERR flag in the SPIx_SR register is set if the value received in the shift register does not match the receiver SPIx_RXCRCR value. The flag is cleared by the software.

TI mode frame format error (FRE)

A TI mode frame format error is detected when an NSS pulse occurs during an ongoing communication when the SPI is operating in slave mode and configured to conform to the TI mode protocol. When this error occurs, the FRE flag is set in the SPIx_SR register. The SPI is not disabled when an error occurs, the NSS pulse is ignored, and the SPI waits for the next NSS pulse before starting a new transfer. The data may be corrupted since the error detection may result in the loss of two data bytes.
The FRE flag is cleared when SPIx_SR register is read. If the ERRIE bit is set, an interrupt is generated on the NSS error detection. In this case, the SPI should be disabled because data consistency is no longer guaranteed and communications should be reinitiated by the master when the slave SPI is enabled again.

24.4.12 NSS pulse mode

This mode is activated by the NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored). When activated, an NSS pulse is generated between two consecutive data frame transfers when NSS stays at high level for the duration of one clock period at least. This mode allows the slave to latch data. NSSP pulse mode is designed for applications with a single master-slave pair.

Figure 263 illustrates NSS pin management when NSSP pulse mode is enabled.

**Figure 263. NSSP pulse generation in Motorola SPI master mode**

![NSSP pulse generation in Motorola SPI master mode](image)

**Note:** Similar behavior is encountered when CPOL = 0. In this case the sampling edge is the rising edge of SCK, and NSS assertion and deassertion refer to this sampling edge.

24.4.13 TI mode

**TI protocol in master mode**

The SPI interface is compatible with the TI protocol. The FRF bit of the SPIx_CR2 register can be used to configure the SPI to be compliant with this protocol.

The clock polarity and phase are forced to conform to the TI protocol requirements whatever the values set in the SPIx_CR1 register. NSS management is also specific to the TI protocol which makes the configuration of NSS management through the SPIx_CR1 and SPIx_CR2 registers (SSM, SSI, SSOE) impossible in this case.

In slave mode, the SPI baud rate prescaler is used to control the moment when the MISO pin state changes to HiZ when the current transaction finishes (see Figure 264). Any baud rate can be used, making it possible to determine this moment with optimal flexibility. However, the baud rate is generally set to the external master clock baud rate. The delay for the MISO signal to become HiZ ($t_{release}$) depends on internal resynchronization and on the
baud rate value set in through the BR[2:0] bits in the SPIx_CR1 register. It is given by the formula:

\[ \frac{t_{\text{baud rate}}}{2} + 4 \times t_{\text{pclk}} < t_{\text{release}} < \frac{t_{\text{baud rate}}}{2} + 6 \times t_{\text{pclk}} \]

If the slave detects a misplaced NSS pulse during a data frame transaction the TIFRE flag is set.

If the data size is equal to 4-bits or 5-bits, the master in full-duplex mode or transmit-only mode uses a protocol with one more dummy data bit added after LSB. TI NSS pulse is generated above this dummy bit clock cycle instead of the LSB in each period.

This feature is not available for Motorola SPI communications (FRF bit set to 0).

*Figure 264: TI mode transfer* shows the SPI communication waveforms when TI mode is selected.

**Figure 264. TI mode transfer**

24.4.14 CRC calculation

Two separate CRC calculators are implemented in order to check the reliability of transmitted and received data. The SPI offers CRC8 or CRC16 calculation independently of the frame data length, which can be fixed to 8-bit or 16-bit. For all the other data frame lengths, no CRC is available.

**CRC principle**

CRC calculation is enabled by setting the CRCEN bit in the SPIx_CR1 register before the SPI is enabled (SPE = 1). The CRC value is calculated using an odd programmable polynomial on each bit. The calculation is processed on the sampling clock edge defined by the CPHA and CPOL bits in the SPIx_CR1 register. The calculated CRC value is checked automatically at the end of the data block as well as for transfer managed by CPU or by the DMA. When a mismatch is detected between the CRC calculated internally on the received data and the CRC sent by the transmitter, a CRCERR flag is set to indicate a data corruption error. The right procedure for handling the CRC calculation depends on the SPI configuration and the chosen transfer management.
Note: The polynomial value should only be odd. No even values are supported.

CRC transfer managed by CPU

Communication starts and continues normally until the last data frame has to be sent or received in the SPIx_DR register. Then CRCNEXT bit has to be set in the SPIx_CR1 register to indicate that the CRC frame transaction follows after the transaction of the currently processed data frame. The CRCNEXT bit must be set before the end of the last data frame transaction. CRC calculation is frozen during CRC transaction.

The received CRC is stored in the RXFIFO like a data byte or word. That is why in CRC mode only, the reception buffer has to be considered as a single 16-bit buffer used to receive only one data frame at a time.

A CRC-format transaction usually takes one more data frame to communicate at the end of data sequence. However, when setting an 8-bit data frame checked by 16-bit CRC, two more frames are necessary to complete the CRC.

When the last CRC data is received, an automatic check is performed comparing the received value and the value in the SPIx_RXCRC register. Software has to check the CRCERR flag in the SPIx_SR register to determine if the data transfers were corrupted or not. Software clears the CRCERR flag by writing '0' to it.

After the CRC reception, the CRC value is stored in the RXFIFO and must be read in the SPIx_DR register in order to clear the RXNE flag.

CRC transfer managed by DMA

When SPI communication is enabled with CRC communication and DMA mode, the transmission and reception of the CRC at the end of communication is automatic (with the exception of reading CRC data in receive only mode). The CRCNEXT bit does not have to be handled by the software. The counter for the SPI transmission DMA channel has to be set to the number of data frames to transmit excluding the CRC frame. On the receiver side, the received CRC value is handled automatically by DMA at the end of the transaction but user must take care to flush out received CRC information from RXFIFO as it is always loaded into it. In full-duplex mode, the counter of the reception DMA channel can be set to the number of data frames to receive including the CRC, which means, for example, in the specific case of an 8-bit data frame checked by 16-bit CRC:

\[ \text{DMA}_\text{RX} = \text{Numb}_\text{of}_\text{data} + 2 \]

In receive only mode, the DMA reception channel counter should contain only the amount of data transferred, excluding the CRC calculation. Then based on the complete transfer from DMA, all the CRC values must be read back by software from FIFO as it works as a single buffer in this mode.

At the end of the data and CRC transfers, the CRCERR flag in the SPIx_SR register is set if corruption occurred during the transfer.

If packing mode is used, the LDMA_RX bit needs managing if the number of data is odd.

Resetting the SPIx_TXCRC and SPIx_RXCRC values

The SPIx_TXCRC and SPIx_RXCRC values are cleared automatically when new data is sampled after a CRC phase. This allows the use of DMA circular mode (not available in receive-only mode) in order to transfer data without any interruption, (several data blocks covered by intermediate CRC checking phases).
If the SPI is disabled during a communication the following sequence must be followed:
1. Disable the SPI
2. Clear the CRCEN bit
3. Enable the CRCEN bit
4. Enable the SPI

**Note:** When the SPI interface is configured as a slave, the NSS internal signal needs to be kept low during transaction of the CRC phase once the CRCNEXT signal is released. That is why the CRC calculation cannot be used at NSS Pulse mode when NSS hardware mode should be applied at slave normally.

At TI mode, despite the fact that clock phase and clock polarity setting is fixed and independent on SPIx_CR1 register, the corresponding setting CPOL=0 CPHA=1 has to be kept at the SPIx_CR1 register anyway if CRC is applied. In addition, the CRC calculation has to be reset between sessions by SPI disable sequence with re-enable the CRCEN bit described above at both master and slave side, else CRC calculation can be corrupted at this specific mode.

### 24.5 SPI interrupts

During SPI communication an interrupt can be generated by the following events:
- Transmit TXFIFO ready to be loaded
- Data received in Receive RXFIFO
- Master mode fault
- Overrun error
- TI frame format error
- CRC protocol error

Interrupts can be enabled and disabled separately.

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable Control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit TXFIFO ready to be loaded</td>
<td>TXE</td>
<td>TXEIE</td>
</tr>
<tr>
<td>Data received in RXFIFO</td>
<td>RXNE</td>
<td>RXNEIE</td>
</tr>
<tr>
<td>Master Mode fault event</td>
<td>MODF</td>
<td></td>
</tr>
<tr>
<td>Overrun error</td>
<td>OVR</td>
<td>ERRIE</td>
</tr>
<tr>
<td>TI frame format error</td>
<td>FRE</td>
<td></td>
</tr>
<tr>
<td>CRC protocol error</td>
<td>CRCERR</td>
<td></td>
</tr>
</tbody>
</table>

For code example refer to the Appendix section A.14.4: SPI interrupt.
24.6 SPI registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit). SPI_DR in addition can be accessed by 8-bit access.

24.6.1 SPI control register 1 (SPIx_CR1)

Address offset: 0x00
Reset value: 0x0000

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BIDI MODE</td>
<td>BIDIOE</td>
<td>CRC EN</td>
<td>CRCN EXT</td>
<td>CRCL</td>
<td>RX ONLY</td>
<td>SSM</td>
<td>SSI</td>
<td>LSB FIRST</td>
<td>SPE</td>
<td>BR[2:0]</td>
<td>MSTR</td>
<td>CPOL</td>
<td>CPHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Bit 15 BIDI MODE: Bidirectional data mode enable.
This bit enables half-duplex communication using common single bidirectional data line.
Keep RXONLY bit clear when bidirectional mode is active.
0: 2-line unidirectional data mode selected
1: 1-line bidirectional data mode selected

Note:

Bit 14 BIDIOE: Output enable in bidirectional mode
This bit combined with the BIDI MODE bit selects the direction of transfer in bidirectional mode.
0: Output disabled (receive-only mode)
1: Output enabled (transmit-only mode)

Note: In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.

Bit 13 CRC EN: Hardware CRC calculation enable
0: CRC calculation disabled
1: CRC calculation enabled

Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.

Bit 12 CRC NEXT: Transmit CRC next
0: Next transmit value is from Tx buffer.
1: Next transmit value is from Tx CRC register.

Note: This bit has to be written as soon as the last data is written in the SPIx_DR register.

Bit 11 CRCL: CRC length
This bit is set and cleared by software to select the CRC length.
0: 8-bit CRC length
1: 16-bit CRC length

Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.
Bit 10  **RXONLY:** Receive only mode enabled.
This bit enables simplex communication using a single unidirectional line to receive data exclusively. Keep BIDIMODE bit clear when receive only mode is active. 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)

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:* This bit is not used in SPI TI 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:* This bit is not used in SPI TI mode.

Bit 7  **LSBFIRST:** Frame format
0: data is transmitted / received with the MSB first
1: data is transmitted / received with the LSB first
*Note:* 1. This bit should not be changed when communication is ongoing.
2. This bit is not used in SPI TI mode.

Bit 6  **SPE:** SPI enable
0: Peripheral disabled
1: Peripheral enabled
*Note:* When disabling the SPI, follow the procedure described in Procedure for disabling the SPI on page 649.

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
*Note:* These bits should not be changed when communication is ongoing.

Bit 2  **MSTR:** Master selection
0: Slave configuration
1: Master configuration
*Note:* This bit should not be changed when communication is ongoing.
Bit 1  **CPOL**: Clock polarity

0: CK to 0 when idle
1: CK to 1 when idle

*Note*: This bit should not be changed when communication is ongoing.

This bit is not used in SPI TI mode except the case when CRC is applied at TI 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

*Note*: This bit should not be changed when communication is ongoing.

This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.

### 24.6.2  SPI control register 2 (SPIx_CR2)

Address offset: 0x04

Reset value: 0x0700

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDMA</td>
<td>LDMA</td>
<td>FRXTH</td>
<td>DS[3:0]</td>
<td>TXEIE</td>
<td>RXNEIE</td>
<td>ERRIE</td>
<td>FRF</td>
<td>NSSP</td>
<td>SSOE</td>
<td>TXDMAEN</td>
<td>RXDMAEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_TX</td>
<td>_RX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bit 15  **Reserved, must be kept at reset value.**

Bit 14  **LDMA_TX**: Last DMA transfer for transmission

This bit is used in data packing mode, to define if the total number of data to transmit by DMA is odd or even. It has significance only if the TXDMAEN bit in the SPIx_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx_CR1 register).

0: Number of data to transfer is even
1: Number of data to transfer is odd

*Note*: Refer to Procedure for disabling the SPI on page 649 if the CRCEN bit is set.

Bit 13  **LDMA_RX**: Last DMA transfer for reception

This bit is used in data packing mode, to define if the total number of data to receive by DMA is odd or even. It has significance only if the RXDMAEN bit in the SPIx_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx_CR1 register).

0: Number of data to transfer is even
1: Number of data to transfer is odd

*Note*: Refer to Procedure for disabling the SPI on page 649 if the CRCEN bit is set.

Bit 12  **FRXTH**: FIFO reception threshold

This bit is used to set the threshold of the RXFIFO that triggers an RXNE event

0: RXNE event is generated if the FIFO level is greater than or equal to 1/2 (16-bit)
1: RXNE event is generated if the FIFO level is greater than or equal to 1/4 (8-bit)
Bits 11:8  **DS[3:0]**: Data size

These bits configure the data length for SPI transfers.

- 0000: Not used
- 0001: Not used
- 0010: Not used
- 0011: 4-bit
- 0100: 5-bit
- 0101: 6-bit
- 0110: 7-bit
- 0111: 8-bit
- 1000: 9-bit
- 1001: 10-bit
- 1010: 11-bit
- 1011: 12-bit
- 1100: 13-bit
- 1101: 14-bit
- 1110: 15-bit
- 1111: 16-bit

If software attempts to write one of the “Not used” values, they are forced to the value “0111” (8-bit)

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.

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.

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, FRE at TI mode).

- 0: Error interrupt is masked
- 1: Error interrupt is enabled

Bit 4  **FRF**: Frame format

- 0: SPI Motorola mode
- 1: SPI TI mode

*Note:* This bit must be written only when the SPI is disabled (SPE=0).

Bit 3  **NSSP**: NSS pulse management

This bit is used in master mode only. It allows the SPI to generate an NSS pulse between two consecutive data when doing continuous transfers. In the case of a single data transfer, it forces the NSS pin high level after the transfer.

It has no meaning if CPHA = ‘1’, or FRF = ‘1’.

- 0: No NSS pulse
- 1: NSS pulse generated

*Note:* 1. This bit must be written only when the SPI is disabled (SPE=0).

2. This bit is not used in SPI TI mode.
24.6.3 SPI status register (SPIx_SR)

Address offset: 0x08
Reset value: 0x0002

| Bit 15:13 | Reserved, must be kept at reset value. |
| Bit 12:11 | FTLVL[1:0]: FIFO transmission level |
|           | These bits are set and cleared by hardware. |
|           | 00: FIFO empty |
|           | 01: 1/4 FIFO |
|           | 10: 1/2 FIFO |
|           | 11: FIFO full (considered as FULL when the FIFO threshold is greater than 1/2) |

| Bit 10:9  | FRLVL[1:0]: FIFO reception level |
|          | These bits are set and cleared by hardware. |
|          | 00: FIFO empty |
|          | 01: 1/4 FIFO |
|          | 10: 1/2 FIFO |
|          | 11: FIFO full |

*Note: These bits are not used in SPI receive-only mode while CRC calculation is enabled.*

| Bit 8    | FRE: Frame format error |
|          | This flag is used for SPI in TI slave mode. Refer to Section 24.4.11: SPI error flags. |
|          | This flag is set by hardware and reset when SPIx_SR is read by software. |
|          | 0: No frame format error |
|          | 1: A frame format error occurred |
Bit 7  **BSY**: Busy flag
- 0: SPI not busy
- 1: SPI is busy in communication or Tx buffer is not empty
  This flag is set and cleared by hardware.

*Note: The BSY flag must be used with caution: refer to Section 24.4.10: SPI status flags and Procedure for disabling the SPI on page 649.*

Bit 6  **OVR**: Overrun flag
- 0: No overrun occurred
- 1: Overrun occurred
  This flag is set by hardware and reset by a 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: Mode fault (MODF) on page 658* for the software sequence.

Bit 4  **CRCERR**: CRC error flag
- 0: CRC value received matches the SPIx_RXCRCR value
- 1: CRC value received does not match the SPIx_RXCRCR value

*Note*: This flag is set by hardware and cleared by software writing 0.

Bits 3:2  Reserved, must be kept at reset value.

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

### 24.6.4 SPI data register (SPIx_DR)

**Address offset**: 0x0C

**Reset value**: 0x0000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>DR[15:0]: Data register</td>
</tr>
<tr>
<td>14</td>
<td>Data received or to be transmitted</td>
</tr>
<tr>
<td>13</td>
<td>The data register serves as an interface between the Rx and Tx FIFOs. When the data register is read, RxFIFO is accessed while the write to data register accesses TxFIFO (See Section 24.4.9: Data transmission and reception procedures).</td>
</tr>
<tr>
<td>12</td>
<td>Note: Data is always right-aligned. Unused bits are ignored when writing to the register, and read as zero when the register is read. The Rx threshold setting must always correspond with the read access currently used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
<th>rw</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

668/775  RM0360 Rev 5
24.6.5 SPI CRC polynomial register (SPIx_CRCPR)

Address offset: 0x10
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>CRCPOLY[15:0]: CRC polynomial register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 15:0</td>
<td>This register contains the polynomial for the CRC calculation.</td>
</tr>
<tr>
<td></td>
<td>The CRC polynomial (0x0007) is the reset value of this register. Another polynomial can be configured as required.</td>
</tr>
</tbody>
</table>

Note: The polynomial value should be odd only. No even value is supported.

24.6.6 SPI Rx CRC register (SPIx_RXCRCR)

Address offset: 0x14
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>RXCRC[15:0]: Rx CRC register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 15:0</td>
<td>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 SPIx_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx_CRCPR register.</td>
</tr>
<tr>
<td></td>
<td>Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx_CR1 is cleared). CRC calculation is done based on any CRC8 standard.</td>
</tr>
<tr>
<td></td>
<td>The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx_CR1 register is set). CRC calculation is done based on any CRC16 standard.</td>
</tr>
<tr>
<td></td>
<td>A read to this register when the BSY Flag is set could return an incorrect value.</td>
</tr>
</tbody>
</table>

24.6.7 SPI Tx CRC register (SPIx_TXCRCR)

Address offset: 0x18
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bits 15:0</th>
<th>TXCRC[15:0]: Tx CRC register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 15:0</td>
<td>This register contains the polynomial for the CRC calculation.</td>
</tr>
<tr>
<td></td>
<td>The CRC polynomial (0x0007) is the reset value of this register. Another polynomial can be configured as required.</td>
</tr>
</tbody>
</table>

Note: The polynomial value should be odd only. No even value is supported.
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 SPIx_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx_CR1 register is set). CRC calculation is done based on any CRC16 standard.

* A read to this register when the BSY flag is set could return an incorrect value.*
24.6.8 SPI register map

Table 97 shows the SPI register map and reset values.

Table 97. SPI register map and reset values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register name</th>
<th>Register reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>SPIx_CR1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIDMODE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>BDEIE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>SRE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>CRNEN</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>CRCL</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>RXONLY</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>LSBFIRST</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>SPE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>BR [2:0]</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>CSR</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>MSTR</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>CPOL</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>CPHA</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>reset value</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0x04</td>
<td>SPIx_CR2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDMA_TX</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>LDMA_RX</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>FRXTH</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>DS[3:0]</td>
<td>0000000000000011</td>
</tr>
<tr>
<td></td>
<td>TXEIE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>RXNEIE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>ERRIE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>FRF</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>NSSP</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>SSOE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>TXDMAEN</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>RXDMAEN</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>reset value</td>
<td>0000000000000011</td>
</tr>
<tr>
<td>0x08</td>
<td>SPIx_SR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FTLVL[1:0]</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>FRLVL[1:0]</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>FRE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>BSY</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>OVR</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>MODF</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>CRCERR</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>RXNE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>BSY</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>reset value</td>
<td>0000000000000001</td>
</tr>
<tr>
<td>0x0C</td>
<td>SPIx_DR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DR[15:0]</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0x10</td>
<td>SPIx_CRCPR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRCPOLY[15:0]</td>
<td>0000000000000111</td>
</tr>
<tr>
<td></td>
<td>reset value</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0x14</td>
<td>SPIx_RXCRCR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RXCRC[15:0]</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>reset value</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0x18</td>
<td>SPIx_TXCRCR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TXCRC[15:0]</td>
<td>0000000000000000</td>
</tr>
<tr>
<td></td>
<td>reset value</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

Refer to Section 2.2 on page 37 for the register boundary addresses.
25 **Universal serial bus full-speed device interface (USB)**

This section applies to STM32F070x6 and STM32F070xB devices only.

25.1 **Introduction**

The USB peripheral implements an interface between a full-speed USB 2.0 bus and the APB bus.

USB suspend/resume are supported, which allows to stop the device clocks for low-power consumption.

25.2 **USB main features**

- USB specification version 2.0 full-speed compliant
- Configurable number of endpoints from 1 to 8
- 1024 bytes of dedicated packet buffer memory SRAM
- 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
- USB 2.0 Link Power Management support
- Battery Charging Specification Revision 1.2 support
- USB connect / disconnect capability (controllable embedded pull-up resistor on USB_DP line)

25.3 **USB implementation**

*Table 98* describes the USB implementation in the devices.

<table>
<thead>
<tr>
<th>USB features(1)</th>
<th>STM32F070x6, STM32F070xB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USB</strong></td>
<td></td>
</tr>
<tr>
<td>Number of endpoints</td>
<td>8</td>
</tr>
<tr>
<td>Size of dedicated packet buffer memory SRAM</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>Dedicated packet buffer memory SRAM access scheme</td>
<td>2 x 16 bits / word</td>
</tr>
<tr>
<td>USB 2.0 Link Power Management (LPM) support</td>
<td>X</td>
</tr>
<tr>
<td>Battery Charging Detection (BCD) support</td>
<td>X</td>
</tr>
<tr>
<td>Embedded pull-up resistor on USB_DP line</td>
<td>X</td>
</tr>
</tbody>
</table>
25.4 USB functional description

Figure 265 shows the block diagram of the USB peripheral.

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. This dedicated memory size is 1024 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 wake-up line to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

25.4.1 Description of USB blocks

The USB peripheral implements all the features related to USB interfacing, which include the following blocks:

- **USB Physical Interface (USB PHY):** This block is maintaining the electrical interface to an external USB host. It contains the differential analog transceiver itself, controllable embedded pull-up resistor (connected to USB_DP line) and support for Battery Charging Detection (BCD), multiplexed on same USB_DP and USB_DM lines. The output enable control signal of the analog transceiver (active low) is provided externally on USB_NOE. It can be used to drive some activity LED or to provide information about the actual communication direction to some other circuitry.

- **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 byte 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 endpoints 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 APB bus through an APB 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
  1024 bytes, structured as 512 half-words of 16 bits.

- **Arbiter:** This block accepts memory requests coming from the APB bus and from the
  USB interface. It resolves the conflicts by giving priority to APB 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 APB 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 half-word set addressed by the APB.

- **APB Wrapper:** This provides an interface to the APB for the memory and register. It
  also maps the whole USB peripheral in the APB address space.

- **Interrupt Mapper:** This block is used to select how the possible USB events can
  generate interrupts and map them to the NVIC.

### 25.5 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.

#### 25.5.1 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.
25.5.2 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 ($t_{\text{STARTUP}}$ 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 10 ms 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 acknowledgment. Since the packet buffer memory has to be accessed by the microcontroller also, an arbitration logic takes care of the access conflicts, using half APB 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 operations.
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 APB bus. Different clock configurations are possible where the APB clock frequency can be higher or lower than the USB peripheral one.

**Note:** Due to USB data rate and packet memory interface requirements, the APB clock must have a minimum frequency of 10 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 half-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 25.6.2: 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 25.5.4: Isochronous transfers and Section 25.5.3: Double-buffered endpoints respectively). The relationship between buffer description table entries and packet buffer areas is depicted in Figure 266.

**Figure 266. 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 are 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, the USB peripheral accesses the contents of ADDRn_TX and COUNTn_TX locations inside the 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 byte to be transmitted (Refer to *Structure and usage of packet buffers on page 676*) and starts sending a DATA0 or DATA1 PID according to USB_EPnR bit DTOG_TX. When the PID is completed, the first byte, 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 half-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 half-words. If a transmitted packet is composed of an odd number of bytes, only the lower half of the last half-word accessed is 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 half-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 is 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.
25.5.3 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.
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>
<thead>
<tr>
<th>Buffer flag</th>
<th>‘Transmission’ endpoint</th>
<th>‘Reception’ endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTOG</td>
<td>DTOG_TX (USB_EPnR bit 6)</td>
<td>DTOG_RX (USB_EPnR bit 14)</td>
</tr>
<tr>
<td>SW_BUF</td>
<td>USB_EPnR bit 14</td>
<td>USB_EPnR bit 6</td>
</tr>
</tbody>
</table>

Table 100. Bulk double-buffering memory buffers usage

<table>
<thead>
<tr>
<th>Endpoint type</th>
<th>DTOG</th>
<th>SW_BUF</th>
<th>Packet buffer used by USB peripheral</th>
<th>Packet buffer used by Application Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN 0 1</td>
<td>ADDRn_TX_0 / COUNTn_TX_0</td>
<td>ADDRn_TX_1 / COUNTn_TX_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>ADDRn_TX_1 / COUNTn_TX_1</td>
<td>ADDRn_TX_0 / COUNTn_TX_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>None (1)</td>
<td>ADDRn_TX_0 / COUNTn_TX_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>None (1)</td>
<td>ADDRn_TX_1 / COUNTn_TX_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT 0 1</td>
<td>ADDRn_RX_0 / COUNTn_RX_0</td>
<td>ADDRn_RX_1 / COUNTn_RX_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>ADDRn_RX_1 / COUNTn_RX_1</td>
<td>ADDRn_RX_0 / COUNTn_RX_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>None (1)</td>
<td>ADDRn_RX_0 / COUNTn_RX_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>None (1)</td>
<td>ADDRn_RX_1 / COUNTn_RX_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer description table locations.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 is 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 100 on page 682). 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 actual transfer rate is 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.

25.5.4 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 101.
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.

### 25.5.5 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 2.5 mA. 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 by not sending any traffic on the USB bus for more than 3 ms: since a SOF packet must be sent every 1 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.

Table 101. Isochronous memory buffers usage

<table>
<thead>
<tr>
<th>Endpoint Type</th>
<th>DTOG bit value</th>
<th>Packet buffer used by the USB peripheral</th>
<th>Packet buffer used by the application software</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>0</td>
<td>ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.</td>
<td>ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.</td>
<td>ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.</td>
</tr>
<tr>
<td>OUT</td>
<td>0</td>
<td>ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.</td>
<td>ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.</td>
<td>ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.</td>
</tr>
</tbody>
</table>
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 10 ms 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 70 ns.

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 102, 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>
<thead>
<tr>
<th>[RXDP,RXDM] status</th>
<th>Wake-up event</th>
<th>Required resume software action</th>
</tr>
</thead>
<tbody>
<tr>
<td>“00”</td>
<td>Root reset</td>
<td>None</td>
</tr>
<tr>
<td>“10”</td>
<td>None (noise on bus)</td>
<td>Go back in Suspend mode</td>
</tr>
</tbody>
</table>
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 1 ms and 15 ms (this interval can be timed using ESOF interrupts, occurring with a 1 ms period when the system clock is running at nominal frequency). Once the RESUME bit is clear, the resume sequence is 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.

### Table 102. Resume event detection (continued)

<table>
<thead>
<tr>
<th>[RXDP,RXDM] status</th>
<th>Wake-up event</th>
<th>Required resume software action</th>
</tr>
</thead>
<tbody>
<tr>
<td>“01”</td>
<td>Root resume</td>
<td>None</td>
</tr>
<tr>
<td>“11”</td>
<td>Not allowed (noise on bus)</td>
<td>Go back in Suspend mode</td>
</tr>
</tbody>
</table>
25.6 USB and USB SRAM registers

The USB peripheral registers can be divided into the following groups:

- Common Registers: Interrupt and Control registers
- Endpoint Registers: Endpoint configuration and status

The USB SRAM registers cover:

- Buffer Descriptor Table: Location of packet memory used to locate data buffers (see Section 2.2: Memory organization to find USB SRAM base address).

All register addresses are expressed as offsets with respect to the USB peripheral registers base address, except the buffer descriptor table locations, which starts at the USB SRAM base address offset by the value specified in the USB_BTABLE register.

Refer to Section 1.2 on page 33 for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

25.6.1 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

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CTR</strong></td>
<td><strong>PMAOVR</strong></td>
<td><strong>ERR</strong></td>
<td><strong>WKUP</strong></td>
<td><strong>SUSP</strong></td>
<td><strong>RESET</strong></td>
<td><strong>SOF</strong></td>
<td><strong>ESOF</strong></td>
<td><strong>L1REQ</strong></td>
<td><strong>L1RESU</strong></td>
<td><strong>RE</strong></td>
<td><strong>F</strong></td>
<td><strong>SUSP</strong></td>
<td><strong>LP</strong></td>
<td><strong>MODE</strong></td>
<td><strong>PDW</strong></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

Bit 15 **CTR**: 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 **PMAOVR**: 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 **ERR**: 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 **WKUP**: Wake-up 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.

Bit 7 **L1REQM**: LPM L1 state request interrupt mask  
0: LPM L1 state request (L1REQ) Interrupt disabled.  
1: L1REQ Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 6 Reserved, must be kept at reset value.

Bit 5 **L1RESUME**: LPM L1 Resume request  
The microcontroller can set this bit to send a LPM L1 Resume signal to the host. After the signaling ends, this bit is cleared by hardware.

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 1 ms and no more than 15 ms 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 signaling 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

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

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 is kept high again. If several bits are set simultaneously, only a single interrupt is 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_EPhR 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 is 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. wake-up 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 3 ms, 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 RESETPM 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 1 ms 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 1 ms, but if the device 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.

Bit 7 **L1REQ**: LPM L1 state request

This bit is set by the hardware when LPM command to enter the L1 state is successfully received and acknowledged. This bit is read/write but only ‘0’ can be written and writing ‘1’ has no effect.

Bits 6:5 Reserved, must be kept at reset value.

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

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>RXDP: Receive data + line status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 14</td>
<td>RXDM: Receive data - line status</td>
</tr>
<tr>
<td>Bit 13</td>
<td>LCK: Locked</td>
</tr>
<tr>
<td>Bit 12:11</td>
<td>LSOF[1:0]: Lost SOF</td>
</tr>
<tr>
<td>Bit 10:0</td>
<td>FN[10:0]: Frame number</td>
</tr>
</tbody>
</table>

**USB device address (USB_DADDR)**
Address offset: 0x4C
Reset value: 0x0000
RM0360 Universal serial bus full-speed device interface (USB)

**Buffer table address (USB_BTABLE)**

Address offset: 0x50  
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

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.

**LPM control and status register (USB_LPMCSR)**

Address offset: 0x54  
Reset value: 0x0000

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

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 USB 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 676).

Bits 2:0  Reserved, forced by hardware to 0.
Bits 15:8 Reserved, must be kept at reset value.

Bits 7:4 **BESL[3:0]**: BESL value
These bits contain the BESL value received with last ACKed LPM Token

Bit 3 **REMWAKE**: bRemoteWake value
This bit contains the bRemoteWake value received with last ACKed LPM Token

Bit 2 Reserved

Bit 1 **LPMACK**: LPM Token acknowledge enable
- 0: the valid LPM Token is NYET.
- 1: the valid LPM Token is ACK.
The NYET/ACK is returned only on a successful LPM transaction:
  - No errors in both the EXT token and the LPM token (else ERROR)
  - A valid bLinkState = 0001B (L1) is received (else STALL)

Bit 0 **LPMEN**: LPM support enable
This bit is set by the software to enable the LPM support within the USB device. If this bit is at '0' no LPM transactions are handled.

**Battery charging detector (USB_BCDR)**

Address offset: 0x58
Reset value: 0x0000

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bit 15 **DPPU**: DP pull-up control
This bit is set by software to enable the embedded pull-up on the DP line. Clearing it to '0' can be used to signalize disconnect to the host when needed by the user software.

Bits 14:8 Reserved, must be kept at reset value.

Bit 7 **PS2DET**: DM pull-up detection status
This bit is active only during PD and gives the result of comparison between DM voltage level and $V_{LGC}$ threshold. In normal situation, the DM level should be below this threshold. If it is above, it means that the DM is externally pulled high. This can be caused by connection to a PS2 port (which pulls-up both DP and DM lines) or to some proprietary charger not following the BCD specification.
- 0: Normal port detected (connected to SDP, ACA, CDP or DCP).
- 1: PS2 port or proprietary charger detected.

Bit 6 **SDET**: Secondary detection (SD) status
This bit gives the result of SD.
- 0: CDP detected.
- 1: DCP detected.

Bit 5 **PDET**: Primary detection (PD) status
This bit gives the result of PD.
- 0: no BCD support detected (connected to SDP or proprietary device).
- 1: BCD support detected (connected to ACA, CDP or DCP).
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>CTR_RX</td>
<td>rw</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>t</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DTOG_RX</td>
<td>rw</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>STAT_RX[1:0]</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SETUP</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>EP_TYPE[1:0]</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>EP_KIND</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CTR_TX</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>DTOG_TX</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>STAT_TX[1:0]</td>
<td>rw</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rc_w0</td>
<td>r</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>EA[3:0]</td>
<td>r</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 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 25.5.3: 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 25.5.4: 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 103: Reception status encoding on page 698. 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 25.5.3: 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 104: Endpoint type encoding on page 698. 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 is 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 25.5.4: 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 105 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 25.5.3: 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 25.5.3: 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 25.5.4: 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 106. 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 an 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 25.5.3: 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 103. Reception status encoding**

<table>
<thead>
<tr>
<th>STAT_RX[1:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>DISABLED: all reception requests addressed to this endpoint are ignored.</td>
</tr>
<tr>
<td>01</td>
<td>STALL: the endpoint is stalled and all reception requests result in a STALL handshake.</td>
</tr>
<tr>
<td>10</td>
<td>NAK: the endpoint is naked and all reception requests result in a NAK handshake.</td>
</tr>
<tr>
<td>11</td>
<td>VALID: this endpoint is enabled for reception.</td>
</tr>
</tbody>
</table>

**Table 104. Endpoint type encoding**

<table>
<thead>
<tr>
<th>EP_TYPE[1:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>BULK</td>
</tr>
<tr>
<td>01</td>
<td>CONTROL</td>
</tr>
<tr>
<td>10</td>
<td>ISO</td>
</tr>
<tr>
<td>11</td>
<td>INTERRUPT</td>
</tr>
</tbody>
</table>

**Table 105. Endpoint kind meaning**

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>BULK</td>
</tr>
<tr>
<td>01</td>
<td>CONTROL</td>
</tr>
<tr>
<td>10</td>
<td>ISO</td>
</tr>
<tr>
<td>11</td>
<td>INTERRUPT</td>
</tr>
</tbody>
</table>
Table 106. Transmission status encoding

<table>
<thead>
<tr>
<th>STAT_TX[1:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td><strong>DISABLED</strong>: all transmission requests addressed to this endpoint are ignored.</td>
</tr>
<tr>
<td>01</td>
<td><strong>STALL</strong>: the endpoint is stalled and all transmission requests result in a STALL handshake.</td>
</tr>
<tr>
<td>10</td>
<td><strong>NAK</strong>: the endpoint is naked and all transmission requests result in a NAK handshake.</td>
</tr>
<tr>
<td>11</td>
<td><strong>VALID</strong>: this endpoint is enabled for transmission.</td>
</tr>
</tbody>
</table>
25.6.2 Buffer descriptor table

**Note:** The buffer descriptor table is located inside the packet buffer memory in the separate "USB SRAM" address space.

Although the buffer descriptor table is located inside the packet buffer memory ("USB SRAM" area), 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 device.

The first packet memory location is located at USB SRAM base address. The buffer descriptor table entry associated with the USB_EPnR registers is described below. The packet memory should be accessed only by byte (8-bit) or half-word (16-bit) accesses. Word (32-bit) accesses are not allowed.

A thorough explanation of packet buffers and the buffer descriptor table usage can be found in *Structure and usage of packet buffers on page 676*.

**Transmission buffer address n (USB_ADDRn_TX)**

Address offset: [USB_BTABLE] + n*8

**Note:** In case of double-buffered or isochronous endpoints in the IN direction, this address location is referred to as USB_ADDRn_TX_0.

In case of double-buffered or isochronous endpoints in the OUT direction, this address location is used for USB_ADDRn_RX_0.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:1</td>
<td>ADDRn_TX[15:1]</td>
</tr>
<tr>
<td>9:0</td>
<td>COUNTn_TX[9:0]</td>
</tr>
</tbody>
</table>

**Transmission byte count n (USB_COUNTn_TX)**

Address offset: [USB_BTABLE] + n*8 + 2

**Note:** In case of double-buffered or isochronous endpoints in the IN direction, this address location is referred to as USB_COUNTn_TX_0.

In case of double-buffered or isochronous endpoints in the OUT direction, this address location is used for USB_COUNTn_RX_0.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:0</td>
<td>COUNTn_TX[9:0]</td>
</tr>
</tbody>
</table>
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.

**Reception buffer address n (USB_ADDRn_RX)**
Address offset: [USB_BTABLE] + n*8 + 4

**Note:**
- **In case of double-buffered or isochronous endpoints in the OUT direction**, this address location is referred to as **USB_ADDRn_RX_1**.
- **In case of double-buffered or isochronous endpoints in the IN direction**, this address location is used for **USB_ADDRn_TX_1**.

<table>
<thead>
<tr>
<th></th>
<th>ADDRn_RX[15:8]</th>
<th>ADDRn_RX[7:1]</th>
<th>ADDRn_RX[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>

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 half-word wide and all packet buffers must be half-word aligned.

**Reception byte count n (USB_COUNTn_RX)**
Address offset: [USB_BTABLE] + n*8 + 6

**Note:**
- **In case of double-buffered or isochronous endpoints in the OUT direction**, this address location is referred to as **USB_COUNTn_RX_1**.
- **In case of double-buffered or isochronous endpoints in the IN direction**, this address location is used for **USB_COUNTn_TX_1**.

<table>
<thead>
<tr>
<th></th>
<th>COUNTn_RX[15:8]</th>
<th>COUNTn_RX[7:1]</th>
<th>COUNTn_RX[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>

This table location is used to store two different values, both required during packet reception. The most significant bits contain 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 half-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 theoretically ranges from 32 to 1024 bytes, which is the longest packet size allowed by USB standard specifications. However, the applicable size is limited by the available buffer memory.

**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 107**.

**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.

**Table 107. Definition of allocated buffer memory**

<table>
<thead>
<tr>
<th>Value of NUM_BLOCK[4:0]</th>
<th>Memory allocated when BL_SIZE=0</th>
<th>Memory allocated when BL_SIZE=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ('00000)</td>
<td>Not allowed</td>
<td>32 bytes</td>
</tr>
<tr>
<td>1 ('00001)</td>
<td>2 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>2 ('00010)</td>
<td>4 bytes</td>
<td>96 bytes</td>
</tr>
<tr>
<td>3 ('00011)</td>
<td>6 bytes</td>
<td>128 bytes</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14 ('01110)</td>
<td>28 bytes</td>
<td>480 bytes</td>
</tr>
<tr>
<td>15 ('01111)</td>
<td>30 bytes</td>
<td>512 bytes</td>
</tr>
<tr>
<td>16 ('10000)</td>
<td>32 bytes</td>
<td>544 bytes</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>29 ('11101)</td>
<td>58 bytes</td>
<td>960 bytes</td>
</tr>
<tr>
<td>30 ('11110)</td>
<td>60 bytes</td>
<td>992 bytes</td>
</tr>
<tr>
<td>31 ('11111)</td>
<td>62 bytes</td>
<td>N/A</td>
</tr>
</tbody>
</table>
25.6.3 USB register map

The table below provides the 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  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x04   | USB_EP1R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x08   | USB_EP2R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C   | USB_EP3R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x10   | USB_EP4R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x14   | USB_EP5R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x18   | USB_EP6R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x1C   | USB_EP7R  | 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  | 0  | 0  | 0  | 0  |
|        | Reset value |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x20   |           | 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  | 0  | 0  | 0  | 0  |

Table 108. USB register map and reset values
Table 108. 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 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x54   | USB_LPMCSR |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x58   | USB_BCDR   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Refer to Section 2.2 on page 37 for the register boundary addresses.
26  Debug support (DBG)

26.1  Overview

The STM32F0x0 devices are built around a Arm® Cortex®-M0 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 STM32F0x0 MCUs.

One interface for debug is available:

- Serial wire

Figure 267. Block diagram of STM32F0x0 MCU and Arm® Cortex®-M0-level debug support

1. The debug features embedded in the Arm® Cortex®-M0 core are a subset of the Arm CoreSight Design Kit.
The Arm® Cortex®-M0 core provides integrated on-chip debug support. It is comprised of:

- SW-DP: Serial wire
- BPU: Break point unit
- DWT: Data watchpoint trigger

It also includes debug features dedicated to the STM32F0x0:

- 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®-M0 core, refer to the Arm® Cortex®-M0 Technical Reference Manual (see [Section 26.2: Reference Arm documentation](#)).

### 26.2 Reference Arm documentation

- Arm® Cortex®-M0 Technical Reference Manual (TRM)
  
  It is available from:  
  [http://infocenter.arm.com](http://infocenter.arm.com)

- Arm Debug Interface V5
- Arm CoreSight Design Kit revision r1p1 Technical Reference Manual

### 26.3 Pinout and debug port pins

The STM32F0x0 MCUs are available in various packages with different numbers of available pins.
26.3.1 SWD port pins

Two pins are used as outputs for the SW-DP as alternate functions of general purpose I/Os. These pins are available on all packages.

<table>
<thead>
<tr>
<th>SW-DP pin name</th>
<th>SW debug port Type</th>
<th>Debug assignment</th>
<th>Pin assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWDIO</td>
<td>IO</td>
<td>Serial Wire Data Input/Output</td>
<td>PA13</td>
</tr>
<tr>
<td>SWCLK</td>
<td>I</td>
<td>Serial Wire Clock</td>
<td>PA14</td>
</tr>
</tbody>
</table>

26.3.2 SW-DP pin assignment

After reset (SYSRESETn or PORESETn), the pins used for the SW-DP are assigned as dedicated pins, immediately usable by the debugger host.

However, the MCU offers the possibility to disable the SWD port and can then release the associated pins for general-purpose I/O (GPIO) usage. For more details on how to disable SW-DP port pins, refer to Section 8.3.2: I/O pin alternate function multiplexer and mapping on page 127.

26.3.3 Internal pull-up and pull-down on SWD pins

Once the SW I/O is released by the user software, the GPIO controller takes control of these pins. The reset states of the GPIO control registers put the I/Os in the equivalent states:

- SWDIO: input pull-up
- SWCLK: input pull-down

Having embedded pull-up and pull-down resistors removes the need to add external resistors.

26.4 ID codes and locking mechanism

There are several ID codes inside the MCU. ST strongly recommends the tool manufacturers (for example Keil, IAR, Raisonance) to lock their debugger using the MCU device ID located at address 0x40015800.

Only the DEV_ID[15:0] should be used for identification by the debugger/programmer tools (the revision ID must not be taken into account).
26.4.1 MCU device ID code

The STM32F0xx products integrate an MCU ID code. This ID identifies the ST MCU part number and the die revision.

This code is accessible by the software debug port (two pins) or by the user software.

For code example refer to the Appendix section A.10.1: DBG read device ID.

**DBGMCU_IDCODE**

Address: 0x40015800

Only 32-bit access supported. Read-only

<table>
<thead>
<tr>
<th>Bits 31:16 REV_ID[15:0] Revision identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>This field indicates the revision of the device. Refer to Table 115.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits 15:12 Reserved: read 0b0110.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Bits 11:0 DEV_ID[11:0]: Device identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>This field indicates the device ID. Refer to Table 115.</td>
</tr>
</tbody>
</table>

**Table 110. DEV_ID and REV_ID field values**

<table>
<thead>
<tr>
<th>Device</th>
<th>DEV_ID</th>
<th>Revision code</th>
<th>Revision number</th>
<th>REV_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F030x4</td>
<td>0x444</td>
<td>A or 1</td>
<td>1.0</td>
<td>0x1000</td>
</tr>
<tr>
<td>STM32F030x6</td>
<td>0x445</td>
<td>A</td>
<td>1.0</td>
<td>0x1000</td>
</tr>
<tr>
<td>STM32F070x6</td>
<td>0x440</td>
<td>B or 1</td>
<td>1.1</td>
<td>0x1001</td>
</tr>
<tr>
<td>STM32F030x8</td>
<td></td>
<td>Z</td>
<td>1.2</td>
<td>0x1003</td>
</tr>
<tr>
<td>STM32F070xB</td>
<td>0x448</td>
<td>Y or 1</td>
<td>2.1</td>
<td>0x2001</td>
</tr>
<tr>
<td>STM32F030xC</td>
<td>0x442</td>
<td>A</td>
<td>1.0</td>
<td>0x1000</td>
</tr>
</tbody>
</table>

26.5 SWD port

26.5.1 SWD 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.

26.5.2 SWD 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>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Start</td>
<td>Must be “1”</td>
</tr>
</tbody>
</table>
| 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 115 on page 712) |
| 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 Arm® Cortex®-M0 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>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.
The DATA transfer must be followed by a turnaround time only if it is a READ transaction.

### 26.5.3 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 to \texttt{0x0BB11477} (corresponding to Arm® Cortex®-M0).

*Note:* \textit{Note that the SW-DP state machine is inactive until the target reads this ID code.}

- The SW-DP state machine is in \texttt{RESET STATE} either after power-on reset, or after the line is high for more than 50 cycles.
- The SW-DP state machine is in \texttt{IDLE STATE} if the line is low for at least two cycles after \texttt{RESET} state.
- After \texttt{RESET} state, it is \texttt{mandatory} to first enter into an \texttt{IDLE state} \texttt{AND} to perform a \texttt{READ access} of the DP-SW ID CODE register. Otherwise, the target will issue a \texttt{FAULT acknowledge response} on another transactions.

Further details of the SW-DP state machine can be found in the \textit{Arm® Cortex®-M0 TRM} and the \textit{CoreSight Design Kit r1p0 TRM}.

### 26.5.4 DP and AP read/write accesses

- Read accesses to the DP are not posted: the target response can be immediate (if \texttt{ACK=OK}) or can be delayed (if \texttt{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.

---

### Table 113. DATA transfer (33 bits)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..31</td>
<td>WDATA or RDATA</td>
<td>Write or Read data</td>
</tr>
<tr>
<td>32</td>
<td>Parity</td>
<td>Single parity of the 32 data bits</td>
</tr>
</tbody>
</table>
26.5.5 SW-DP registers

Access to these registers are initiated when APnDP=0

Table 114. SW-DP registers

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Read</td>
<td></td>
<td>IDCODE</td>
<td>The manufacturer code is set to the default Arm code for Cortex-M0: 0x0BB11477 (identifies the SW-DP)</td>
</tr>
<tr>
<td>00</td>
<td>Write</td>
<td></td>
<td>ABORT</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Read/Write</td>
<td>0</td>
<td>DP-CTRL/STAT</td>
<td>Purpose is to:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– request a system or debug power-up</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– configure the transfer operation for AP accesses</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– control the pushed compare and pushed verify operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>– read some status flags (overrun, power-up acknowledges)</td>
</tr>
<tr>
<td>01</td>
<td>Read/Write</td>
<td>1</td>
<td>WIRE CONTROL</td>
<td>Purpose is to configure the physical serial port protocol (like the duration of the turnaround time)</td>
</tr>
<tr>
<td>10</td>
<td>Read</td>
<td></td>
<td>READ RESEND</td>
<td>Enables recovery of the read data from a corrupted debugger transfer, without repeating the original AP transfer.</td>
</tr>
<tr>
<td>10</td>
<td>Write</td>
<td></td>
<td>SELECT</td>
<td>The purpose is to select the current access port and the active 4-words register window</td>
</tr>
<tr>
<td>11</td>
<td>Read/Write</td>
<td></td>
<td>READ BUFFER</td>
<td>This read buffer is useful because AP accesses are posted (the result of a read AP request is available on the next AP transaction).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This read buffer captures data from the AP, presented as the result of a previous read, without initiating a new transaction</td>
</tr>
</tbody>
</table>
26.5.6 SW-AP registers

Access to these registers are initiated when APnDP=1

There are many AP Registers addressed as the combination of:
- The shifted value A[3:2]
- The current value of the DP SELECT register.

Table 115. 32-bit debug port registers addressed through the shifted value A[3:2]

<table>
<thead>
<tr>
<th>Address</th>
<th>A[3:2] value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>00</td>
<td>Reserved, must be kept at reset value.</td>
</tr>
</tbody>
</table>
| 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) |

26.6 Core debug

Core debug is accessed through the core debug registers. Debug access to these registers is by means of the debug access port. It consists of four registers:

Table 116. Core debug registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
These registers are not reset by a system reset. They are only reset by a power-on reset. Refer to the Arm® Cortex®-M0 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

### 26.7 BPU (Break Point Unit)

The Cortex-M0 BPU implementation provides four breakpoint registers. The BPU is a subset of the Flash Patch and Breakpoint (FPB) block available in Armv7-M (Cortex-M3 & Cortex-M4).

#### 26.7.1 BPU functionality

The processor breakpoints implement PC based breakpoint functionality.

Refer to the Armv6-M Arm and the Arm CoreSight Components Technical Reference Manual for more information about the BPU CoreSight identification registers, and their addresses and access types.

### 26.8 DWT (Data Watchpoint)

The Cortex-M0 DWT implementation provides two watchpoint register sets.

#### 26.8.1 DWT functionality

The processor watchpoints implement both data address and PC based watchpoint functionality, a PC sampling register, and support comparator address masking, as described in the Armv6-M Arm.

#### 26.8.2 DWT Program Counter Sample Register

A processor that implements the data watchpoint unit also implements the Armv6-M optional DWT Program Counter Sample Register (DWT_PCSR). This register permits a debugger to periodically sample the PC without halting the processor. This provides coarse grained profiling. See the ARMv6-M Arm for more information.

The Cortex-M0 DWT_PCSR records both instructions that pass their condition codes and those that fail.

### 26.9 MCU debug component (DBGMCU)

The MCU debug component helps the debugger provide support for:

- Low-power modes
- Clock control for timers, watchdog and I2C during a breakpoint
26.9.1 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: FCLK and HCLK are still active. Consequently, this mode does not impose any restrictions on the standard debug features.
- In Stop/Standby mode, the DBG_STOP bit must be previously set by the debugger. This enables the internal RC oscillator clock to feed FCLK and HCLK in Stop mode.

For code example refer to the Appendix section A.10.2: DBG debug in Low-power mode.

26.9.2 Debug support for timers, watchdog and I²C

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 I²C, the user can choose to block the SMBUS timeout during a breakpoint.
### 26.9.3 Debug MCU configuration register (DBGMCU_CR)

This register allows the configuration of the MCU under DEBUG. This concerns:

- Low-power mode support

This DBGMCU_CR is mapped at address 0x40015804.

It is asynchronously reset by the PORRESET (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.

Address: 0x40015804

Only 32-bit access supported

POR Reset: 0x00000000 (not reset by system reset)

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:3 Reserved, must be kept at reset value.

**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)
26.9.4 Debug MCU APB1 freeze register (DBGMCU_APB1_FZ)

The DBGMCU_APB1_FZ register is used to configure the MCU under DEBUG. It concerns some APB peripherals:

- Timer clock counter freeze
- I2C SMBUS timeout freeze
- System window watchdog and independent watchdog counter freeze support

This DBGMCU_APB1_FZ is mapped at address 0x4001 5808.

The register is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address offset: 0x08

Only 32-bit access are supported.

Power on reset (POR): 0x0000 0000 (not reset by system reset)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
</table>

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **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

Bits 20:13 Reserved, must be kept at reset value.

Bit 12 **DBG_IWDG_STOP**: Debug independent watchdog stopped when core is halted
0: The independent watchdog counter clock continues even if the core is halted
1: The independent watchdog counter clock is stopped when the core is halted

Bit 11 **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 10  **DBG_RTC_STOP**: Debug RTC stopped when core is halted
   0: The clock of the RTC counter is fed even if the core is halted
   1: The clock of the RTC counter is stopped when the core is halted

Bit 9  Reserved, must be kept at reset value.

Bit 8  **DBG_TIM14_STOP**: TIM14 counter stopped when core is halted
   0: The counter clock of TIM14 is fed even if the core is halted
   1: The counter clock of TIM14 is stopped when the core is halted

Bits 7:6  Reserved, must be kept at reset value.

Bit 5  **DBG_TIM7_STOP**: TIM7 counter stopped when core is halted.
   0: The counter clock of TIM7 is fed even if the core is halted
   1: The counter clock of TIM7 is stopped when the core is halted

Bit 4  **DBG_TIM6_STOP**: TIM6 counter stopped when core is halted
   0: The counter clock of TIM6 is fed even if the core is halted
   1: The counter clock of TIM6 is stopped when the core is halted

Bits 3:2  Reserved, must be kept at reset value.

Bit 1  **DBG_TIM3_STOP**: TIM3 counter stopped when core is halted
   0: The counter clock of TIM3 is fed even if the core is halted
   1: The counter clock of TIM3 is stopped when the core is halted

Bit 0  Reserved, must be kept at reset value.
26.9.5  Debug MCU APB2 freeze register (DBGMCU_APB2_FZ)

The DBGMCU_APB2_FZ register is used to configure the MCU under DEBUG. It concerns some APB peripherals:

- Timer clock counter freeze

This register is mapped at address 0x4001580C.

It is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address offset: 0x0C

Only 32-bit access is supported.

POR: 0x0000 0000 (not reset by system reset)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:19  Reserved, must be kept at reset value.

Bit 18  DBG_TIM17_STOP: TIM17 counter stopped when core is halted
0: The counter clock of TIM17 is fed even if the core is halted
1: The counter clock of TIM17 is stopped when the core is halted

Bit 17  DBG_TIM16_STOP: TIM16 counter stopped when core is halted
0: The counter clock of TIM16 is fed even if the core is halted
1: The counter clock of TIM16 is stopped when the core is halted

Bit 16  DBG_TIM15_STOP: TIM15 counter stopped when core is halted
0: The counter clock of TIM15 is fed even if the core is halted
1: The counter clock of TIM15 is stopped when the core is halted

Bits 15:12  Reserved, must be kept at reset value.

Bit 11  DBG_TIM1_STOP: TIM1 counter stopped when core is halted
0: The counter clock of TIM1 is fed even if the core is halted
1: The counter clock of TIM1 is stopped when the core is halted

Bits 0:10  Reserved, must be kept at reset value.

26.9.6  DBG register map

The following table summarizes the Debug registers.
### Table 117. DBG register map and reset values

<table>
<thead>
<tr>
<th>Addr.</th>
<th>Register</th>
<th>Addr.</th>
<th>Register</th>
<th>Addr.</th>
<th>Register</th>
<th>Addr.</th>
<th>Register</th>
<th>Addr.</th>
<th>Register</th>
<th>Addr.</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40015800</td>
<td><strong>DBGMCU_IDCODE</strong></td>
<td>21</td>
<td>REV_ID</td>
<td>25</td>
<td>REV_ID</td>
<td>29</td>
<td>REV_ID</td>
<td>31</td>
<td>REV_ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
<td></td>
<td>23</td>
<td></td>
<td>26</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td></td>
<td>21</td>
<td></td>
<td>19</td>
<td></td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>24</td>
<td></td>
<td>25</td>
<td></td>
<td>17</td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
<td></td>
<td>29</td>
<td></td>
<td>15</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td></td>
<td>31</td>
<td></td>
<td>13</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **Reset value**

1. The reset value is product dependent. For more information, refer to *Section 26.4.1: MCU device ID code*. 

---

**Note:** The reset value is product dependent. For more information, refer to *Section 26.4.1: MCU device ID code*. 

---

**ST**
27 Device electronic signature

The device electronic signature is stored in the System memory area of the Flash memory module, and can be read using the debug interface or by the CPU. It contains factory-programmed identification and calibration data that allow the user firmware or other external devices to automatically match to the characteristics of the STM32F0x0 microcontroller.

27.1 Flash memory size data register

Base address: 0x1FFF
Address offset: 0x00
Read only = 0xXXXX where X is factory-programmed

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>f</td>
</tr>
</tbody>
</table>

Bits 15:0 FLASH_SIZE[15:0]: Flash memory size
This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.
Appendix A  Code examples

A.1 Introduction

This appendix shows the code examples of the sequences described in this document. These code examples are extracted from the STM32F0xx Snippet firmware package STM32SnippetsF0 available on www.st.com.

These code examples use the peripheral bit and register description from the CMSIS header file (stm32f0xx.h).

Code lines starting with // should be uncommented if the given register has been modified before.

A.2 FLASH operation code examples

A.2.1 Flash memory unlocking sequence

/* (1) Wait till no operation is on going */
/* (2) Check that the flash memory is unlocked */
/* (3) Perform unlock sequence */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (1) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->CR & FLASH_CR_LOCK) != 0) /* (2) */
{
    FLASH->KEYR = FLASH_FKEY1; /* (3) */
    FLASH->KEYR = FLASH_FKEY2;
}

A.2.2 Main flash memory programming sequence

/* (1) Set the PG bit in the FLASH_CR register to enable programming */
/* (2) Perform the data write (half-word) at the desired address */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) clear it by software by writing it at 1 */
/* (6) Reset the PG Bit to disable programming */
FLASH->CR |= FLASH_CR_PG; /* (1) */
*(__IO uint16_t*)(flash_addr) = data; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH->SR = FLASH_SR_EOP; /* (5) */
} else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_PG; /* (6) */

A.2.3 Page erase sequence

/* (1) Set the PER bit in the FLASH_CR register to enable page erasing */
/* (2) Program the FLASH_AR register to select a page to erase */
/* (3) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (4) Wait until the BSY bit is reset in the FLASH_SR register */
/* (5) Check the EOP flag in the FLASH_SR register */
/* (6) Clear EOP flag by software by writing EOP at 1 */
/* (7) Reset the PER Bit to disable the page erase */
FLASH->CR |= FLASH_CR_PER; /* (1) */
FLASH->AR = page_addr; /* (2) */
FLASH->CR |= FLASH_CR_STRT; /* (3) */
while (((FLASH->SR & FLASH_SR_BSY) != 0) /* (4) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (5) */
{
    FLASH->SR = FLASH_SR_EOP; /* (6) */
} else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_PER; /* (7) */
A.2.4  Mass erase sequence

/* (1) Set the MER bit in the FLASH_CR register to enable mass erasing */
/* (2) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear EOP Flag by software by writing EOP at 1 */
/* (6) Reset the PER Bit to disable the mass erase */
FLASH->CR |= FLASH_CR_MER; /* (1) */
FLASH->CR |= FLASH_CR_STRT; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_MER; /* (6) */

A.2.5  Option byte unlocking sequence

/* (1) Wait till no operation is on going */
/* (2) Check that the flash memory is unlocked */
/* (3) Perform unlock sequence for flash memory */
/* (4) Check that the Option Bytes are unlocked */
/* (5) Perform unlock sequence for Option Bytes */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (1) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->CR & FLASH_CR_LOCK) != 0) /* (2) */
{
    FLASH->KEYR = FLASH_FKEY1; /* (3) */
    FLASH->KEYR = FLASH_FKEY2;
}
if ((FLASH->CR & FLASH_CR_OPTWRE) == 0) /* (4) */
{
    FLASH->OPTKEYR = FLASH_OPTKEY1; /* (5) */
    FLASH->OPTKEYR = FLASH_OPTKEY2;
}
A.2.6  Option byte programming sequence

/* (1) Set the PG bit in the FLASH_CR register to enable programming */
/* (2) Perform the data write */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear the EOP flag by software by writing it at 1 */
/* (6) Reset the PG Bit to disable programming */
FLASH-&gt;CR |= FLASH_CR_OPTPG; /* (1) */
*opt_addr = data; /* (2) */
while ((FLASH-&gt;SR &amp; FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH-&gt;SR &amp; FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH-&gt;SR = FLASH_SR_EOP; /* (5) */
}
else
{
    /* Manage the error cases */
}
FLASH-&gt;CR &= ~FLASH_CR_OPTPG; /* (6) */

A.2.7  Option byte erasing sequence

/* (1) Set the OPTER bit in the FLASH_CR register to enable option byte erasing */
/* (2) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear EOP flag by software by writing EOP at 1 */
/* (6) Reset the PER Bit to disable the page erase */
FLASH-&gt;CR |= FLASH_CR_OPTER; /* (1) */
FLASH-&gt;CR |= FLASH_CR_STRT; /* (2) */
while ((FLASH-&gt;SR &amp; FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH-&gt;SR &amp; FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH-&gt;SR = FLASH_SR_EOP; /* (5)*/
}
else
{
    /* Manage the error cases */
}
FLASH-&gt;CR &= ~FLASH_CR_OPTER; /* (6) */
A.3 Clock controller code examples

A.3.1 HSE start sequence

/**
 * Description: This function enables the interrupt on HSE ready,
 *              and start the HSE as external clock.
 */
__INLINE void StartHSE(void)
{
    /* Configure NVIC for RCC */
    /* (1) Enable Interrupt on RCC */
    /* (2) Set priority for RCC */
    NVIC_EnableIRQ(RCC_CRS_IRQn); /* (1) */
    NVIC_SetPriority(RCC_CRS_IRQn, 0); /* (2) */

    /* (1) Enable interrupt on HSE ready */
    /* (2) Enable the CSS
       Enable the HSE and set HSEBYP to use the external clock
       instead of an oscillator
       Enable HSE */
    /* Note : the clock is switched to HSE in the RCC_CRS_IRQHandler ISR */
    RCC->CIR |= RCC_CIR_HSERDYIE; /* (1) */
    RCC->CR |= RCC_CR_CSSON | RCC_CR_HSEBYP | RCC_CR_HSEON; /* (2) */
}

/**
 * Description: This function handles RCC interrupt request
 *              and switch the system clock to HSE.
 */
void RCC_CRS_IRQHandler(void)
{
    /* (1) Check the flag HSE ready */
    /* (2) Clear the flag HSE ready */
    /* (3) Switch the system clock to HSE */
    if (((RCC->CIR & RCC_CIR_HSERDYF) != 0) /* (1) */) {
        RCC->CIR |= RCC_CIR_HSERDYC; /* (2) */
        RCC->CFGR = ((RCC->CFGR & (~RCC_CFGR_SW)) | RCC_CFGR_SW_0); /* (3) */
    } else {
        /* Report an error */
    }
}
A.3.2 PLL configuration modification

/* (1) Test if PLL is used as System clock */
/* (2) Select HSI as system clock */
/* (3) Wait for HSI switched */
/* (4) Disable the PLL */
/* (5) Wait until PLLRDY is cleared */
/* (6) Set the PLL multiplier to 6 */
/* (7) Enable the PLL */
/* (8) Wait until PLLRDY is set */
/* (9) Select PLL as system clock */
/* (10) Wait until the PLL is switched on */
if ((RCC->CFGR & RCC_CFGR_SWS) == RCC_CFGR_SWS_PLL) /* (1) */
{
    RCC->CFGR &= (uint32_t) (~RCC_CFGR_SW); /* (2) */
    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI) /* (3) */
    {
        /* For robust implementation, add here time-out management */
    }
}
RCC->CR &= (uint32_t) (~RCC_CR_PLLON); /* (4) */
while ((RCC->CR & RCC_CR_PLLRDY) != 0) /* (5) */
{
    /* For robust implementation, add here time-out management */
}
RCC->CFGR = RCC->CFGR & (~RCC_CFGR_PLLMUL) | (RCC_CFGR_PLLMUL6); /* (6) */
RCC->CR |= RCC_CR_PLLON; /* (7) */
while ((RCC->CR & RCC_CR_PLLRDY) == 0) /* (8) */
{
    /* For robust implementation, add here time-out management */
}
RCC->CFGR |= (uint32_t) (RCC_CFGR_SW_PLL); /* (9) */
while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL) /* (10) */
{
    /* For robust implementation, add here time-out management */
}

A.3.3 MCO selection

/* Select system clock to be output on the MCO without prescaler */
RCC->CFGR |= RCC_CFGR_MCO_SYSCLK;
A.3.4 Clock measurement configuration with TIM14

```c
/**
 * Description: This function configures the TIM14 as input capture
 * and enables the interrupt on TIM14
 */
__INLINE void ConfigureTIM14asInputCapture(void)
{
    /* (1) Enable the peripheral clock of Timer 14 */
    /* (2) Select the active input TI1, Program the input filter, and prescaler */
    /* (3) Enable interrupt on Capture/Compare */
    RCC->APB1ENR |= RCC_APB1ENR_TIM14EN; /* (1) */
    TIM14->CCMR1 |= TIM_CCMR1_IC1F_0 | TIM_CCMR1_IC1F_1 \\
        | TIM_CCMR1_CC1S_0 | TIM_CCMR1_IC1PSC_1; /* (2)*/
    TIM14->DIER |= TIM_DIER_CC1IE; /* (3) */

    /* Configure NVIC for TIM14 */
    /* (4) Enable Interrupt on TIM14 */
    /* (5) Set priority for TIM14 */
    NVIC_EnableIRQ(TIM14_IRQn); /* (4) */
    NVIC_SetPriority(TIM14_IRQn, 0); /* (5) */

    /* (6) Select HSE/32 as input on TI1 */
    /* (7) Enable counter */
    /* (8) Enable capture */
    TIM14->OR |= TIM14_OR_TI1_RMP_1; /* (6) */
    TIM14->CR1 |= TIM_CR1_CEN; /* (7) */
    TIM14->CCER |= TIM_CCER_CC1E; /* (8) */
}

Note: The measurement is done in the TIM14 interrupt subroutine.
```
A.4 GPIO code examples

A.4.1 Lock sequence

/**
 * Description: This function locks the targeted pins of Port A configuration
 * This function can be easily modified to lock Port B
 * Parameter: lock contains the port pin mask to be locked
 */
void LockGPIOA(uint16_t lock)
{
    /* (1) Write LCKK bit to 1 and set the pin bits to lock */
    /* (2) Write LCKK bit to 0 and set the pin bits to lock */
    /* (3) Write LCKK bit to 1 and set the pin bits to lock */
    /* (4) Read the Lock register */
    /* (5) Check the Lock register (optionnal) */
    GPIOA->LCKR = GPIO_LCKR_LCKK + lock; /* (1) */
    GPIOA->LCKR = lock; /* (2) */
    GPIOA->LCKR = GPIO_LCKR_LCKK + lock; /* (3) */
    GPIOA->LCKR; /* (4) */
    if (((GPIOA->LCKR & GPIO_LCKR_LCKK) == 0)) /* (5) */
    {
        /* Manage an error */
    }
}

A.4.2 Alternate function selection sequence

/* This sequence select AF2 for GPIOA4, 8 and 9. This can be easily adapted with another port by changing all GPIOA references by another GPIO port, and the alternate function number can be changed by replacing 0x04 or 0x02 for each pin by the targeted alternate function in the 2 last code lines. */
/* (1) Enable the peripheral clock of GPIOA */
/* (2) Select alternate function mode on GPIOA pin 4, 8 and 9 */
/* (3) Select AF4 on PA4 in AFRL for TIM14_CH1 */
/* (4) Select AF2 on PA8 and PA9 in AFRH for TIM1_CH1 and TIM1_CH2 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (1) */
GPIOA->MODER = (GPIOA->MODER & ~ (GPIO_MODER_MODER4 | GPIO_MODER_MODER8 | GPIO_MODER_MODER9)) | GPIO_MODER_MODER4 | GPIO_MODER_MODER8 | GPIO_MODER_MODER9; /* (2) */
GPIOA->AFR[0] |= 0x04 << GPIO_AFRL_AFR4_Pos; /* (3) */
GPIOA->AFR[1] |= (0x02 << GPIO_AFRL_AFR8_Pos) | (0x02 << GPIO_AFRL_AFR9_Pos); /* (4) */
A.4.3  Analog GPIO configuration

/* (1) Enable the peripheral clock of GPIOA, GPIOB and GPIOC */
/* (2) Select analog mode for PA1 */
/* (3) Select analog mode for PB1 */
/* (4) Select analog mode for PC0 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOCEN; /* (1) */
GPIOA->MODER |= GPIO_MODER_MODER1; /* (2) */
GPIOB->MODER |= GPIO_MODER_MODER1; /* (3) */
GPIOC->MODER |= GPIO_MODER_MODER0; /* (4) */

A.5  DMA code examples

A.5.1  DMA channel configuration sequence

/* The following example is given for the ADC. It can be easily ported on any peripheral supporting DMA transfer taking of the associated channel to the peripheral, this must check in the datasheet. */
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performs on channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADCl->CFGR1 |= ADC_CFGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t)(6*ADCl->DR); /* (3) */
DMA1_Channel1->CMAR = (uint32_t)(ADC_array); /* (4) */
DMA1_Channel1->CNDTR = 3; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
| DMA_CCR_TEIE | DMA_CCR_TCIE; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */
/* Configure NVIC for DMA */
/* (1) Enable Interrupt on DMA Channel 1 */
/* (2) Set priority for DMA Channel 1 */
NVIC_EnableIRQ(DMA1_Channel1_IRQn); /* (1) */
NVIC_SetPriority(DMA1_Channel1_IRQn, 0); /* (2) */
A.6 Interrupts and event code examples

A.6.1 NVIC initialization

/* (1) Enable Interrupt on ADC */
/* (2) Set priority for ADC to 2*/
NVIC_EnableIRQ(ADC1_IRQn); /* (1) */
NVIC_SetPriority(ADC1_IRQn,2); /* (2) */

A.6.2 External interrupt selection

/* (1) Enable the peripheral clock of GPIOA */
/* (2) Select Port A for pin 0 external interrupt by writing 0000 in
 EXTI0 (reset value)*/
/* (3) Configure the corresponding mask bit in the EXTI_IMR register*/
/* (4) Configure the Trigger Selection bits of the Interrupt line on
 rising edge*/
/* (5) Configure the Trigger Selection bits of the Interrupt line on
 falling edge*/
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (1) */
//SYSCFG->EXTICR[1] &= (uint16_t)SYSCFG_EXTICR1_EXTI0_PA; /* (2) */
EXTI->IMR = 0x0001; /* (3) */
EXTI->RTSR = 0x0001; /* (4) */
EXTI->FTSR = 0x0001; /* (5) */
/* Configure NVIC for External Interrupt */
/* (1) Enable Interrupt on EXTI0_1 */
/* (2) Set priority for EXTI0_1 */
NVIC_EnableIRQ(EXTI0_1_IRQn); /* (1) */
NVIC_SetPriority(EXTI0_1_IRQn,0); /* (2) */
A.7 ADC code examples

A.7.1 ADC calibration

/* (1) Ensure that ADEN = 0 */
/* (2) Clear ADEN by setting ADDIS */
/* (3) Clear DMAEN */
/* (4) Launch the calibration by setting ADCAL */
/* (5) Wait until ADCAL=0 */
if ((ADC1->CR & ADC_CR_ADEN) != 0) /* (1) */
{ 
  ADC1->CR |= ADC_CR_ADDIS; /* (2) */
}
while ((ADC1->CR & ADC_CR_ADEN) != 0)
{
  /* For robust implementation, add here time-out management */
}
ADC1->CFGR1 &= ~ADC_CFGR1_DMAEN; /* (3) */
ADC1->CR |= ADC_CR_ADCAL; /* (4) */
while ((ADC1->CR & ADC_CR_ADCAL) != 0) /* (5) */
{
  /* For robust implementation, add here time-out management */
}

A.7.2 ADC enable sequence

/* (1) Ensure that ADRDY = 0 */
/* (2) Clear ADRDY */
/* (3) Enable the ADC */
/* (4) Wait until ADC ready */
if ((ADC1->ISR & ADC_ISR_ADRDY) != 0) /* (1) */
{ 
  ADC1->ISR |= ADC_CR_ADRDY; /* (2) */
}
ADC1->CR |= ADC_CR_ADEN; /* (3) */
while ((ADC1->ISR & ADC_ISR_ADRDY) != 0) /* (4) */
{
  /* For robust implementation, add here time-out management */
}
A.7.3 ADC disable sequence

/* (1) Stop any ongoing conversion */
/* (2) Wait until ADSTP is reset by hardware i.e. conversion is stopped */
/* (3) Disable the ADC */
/* (4) Wait until the ADC is fully disabled */
ADC1->CR |= ADC_CR_ADSTP; /* (1) */
while (((ADC1->CR & ADC_CR_ADSTP) != 0)) /* (2) */
{
    /* For robust implementation, add here time-out management */
}
ADC1->CR |= ADC_CR_ADDIS; /* (3) */
while (((ADC1->CR & ADC_CR_ADEN) != 0)) /* (4) */
{
    /* For robust implementation, add here time-out management */
}

A.7.4 ADC clock selection

/* This code selects the HSI14 as clock source. */
/* (1) Enable the peripheral clock of the ADC */
/* (2) Start HSI14 RC oscillator */
/* (3) Wait HSI14 is ready */
/* (4) Select HSI14 by writing 00 in CKMODE (reset value) */
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; /* (1) */
RCC->CR2 |= RCC_CR2_HSI14ON; /* (2) */
while (((RCC->CR2 & RCC_CR2_HSI14RDY) == 0)) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
//ADC1->CFGR2 &= (~ADC_CFGR2_CKMODE); /* (4) */
A.7.5  Single conversion sequence - software trigger

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select CHSEL0, CHSEL9, CHSEL10 and CHSEL17 for VRefInt */
/* (3) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater
than 17.1us */
/* (4) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL9 |
ADC_CHSELR_CHSEL10 | ADC_CHSELR_CHSEL17; /* (2) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (3) */
ADC->CCR |= ADC_CCR_VREFEN; /* (4) */
while (1)
{
    /* Performs the AD conversion */
    ADC1->CR |= ADC_CR_ADSTART; /* Start the ADC conversion */
    for (i=0; i < 4; i++)
    {
        while (((ADC1->ISR & ADC_ISR_EOC) == 0)) /* Wait end of conversion */
        {
            /* For robust implementation, add here time-out management */
        }
        ADC_Result[i] = ADC1->DR; /* Store the ADC conversion result */
    }
    ADC1->CFGR1 ^= ADC_CFGR1_SCANDIR; /* Toggle the scan direction */
}

A.7.6  Continuous conversion sequence - software trigger

/* This code example configures the AD conversion in continuous mode and in
backward scan. It also enable the interrupts. */
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the continuous mode and scanning direction */
/* (3) Select CHSEL1, CHSEL9, CHSEL10 and CHSEL17 */
/* (4) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater than
17.1us */
/* (5) Enable interrupts on EOC, EOSEQ and overrun */
/* (6) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
ADC1->CFGR2 ^= ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_CONT | ADC_CFGR1_SCANDIR; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL9 |
ADC_CHSELR_CHSEL10 | ADC_CHSELR_CHSEL17; /* (3) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (4) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (5) */
ADC->CCR |= ADC_CCR_VREFEN; /* (6) */
/* Configure NVIC for ADC */
/* (7) Enable Interrupt on ADC */
/* (8) Set priority for ADC */
NVIC_EnableIRQ(ADC1_IRQn); /* (7) */
NVIC_SetPriority(ADC1_IRQn, 0); /* (8) */
A.7.7 Single conversion sequence - hardware trigger

/* Configure the ADC, the ADC and its clock having previously been enabled. */
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on falling edge and external trigger on TIM15_TRGO */
/* (3) Select CHSEL0, 1, 2 and 3 */
//ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_0 | ADC_CFGR1_EXTSEL_2; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1
| ADC_CHSELR_CHSEL2 | ADC_CHSELR_CHSEL3; /* (3) */

A.7.8 Continuous conversion sequence - hardware trigger

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO (EXTSEL = 100), falling edge (EXTEN = 10), the continuous mode (CONT = 1) */
/* (3) Select CHSELO/1/2/3 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
//ADC1->CFGR2 |= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_1 | ADC_CFGR1_EXTSEL_2
| ADC_CFGR1_CONT; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1
| ADC_CHSELR_CHSEL2 | ADC_CHSELR_CHSEL3; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

/* Configure NVIC for ADC */
/* (1) Enable Interrupt on ADC */
/* (2) Set priority for ADC */
NVIC_EnableIRQ(ADC1_IRQn); /* (1) */
NVIC_SetPriority(ADC1_IRQn, 0); /* (2) */
A.7.9 DMA one shot mode sequence

/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC - DMACFG is kept at 0
 for one shot mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performed
 on DMA channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t) (ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0 |
DMA_CCR_TEIE | DMA_CCR_TCIE; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */

A.7.10 DMA circular mode sequence

/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC and circular mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performed
 on DMA channel 1 */
/* (6) Configure increment, size, interrupts and circular mode */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN | ADC_CFGR1_DMACFG; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t) (ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0 |
DMA_CCR_TEIE | DMA_CCR_TCIE | DMA_CCR_CIRC; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */

A.7.11 Wait mode sequence

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the continuous mode and the wait mode */
/* (3) Select CHSEL1/2/3 */
ADC1->CFGR2 |= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_CONT | ADC_CFGR1_WAIT; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2 |
ADC_CHSELR_CHSEL3; /* (3) */
ADC1->CR |= ADC_CR_ADSTART; /* start the ADC conversions */
A.7.12 Auto Off and no wait mode sequence

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO and rising edge and auto off */
/* (3) Select CHSEL1/2/3/4 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */

ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_1 | ADC_CFGR1_EXTSEL_2
| ADC_CFGR1_AUTOFF; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
| ADC_CHSELR_CHSEL3 | ADC_CHSELR_CHSEL4; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

A.7.13 Auto Off and wait mode sequence

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO and falling edge, the continuous mode, scanning direction and auto off */
/* (3) Select CHSEL1, CHSEL9, CHSEL10 and CHSEL17 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */

ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_0 | ADC_CFGR1_EXTSEL_2
| ADC_CFGR1_SCANDIR | ADC_CFGR1_AUTOFF; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
| ADC_CHSELR_CHSEL3 | ADC_CHSELR_CHSEL4; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

A.7.14 Analog watchdog

/* (1) Select the continuous mode and configure the Analog watchdog to monitor only CH17 */
/* (2) Define analog watchdog range : 16b-MSW is the high limit and 16b-LSW is the low limit */
/* (3) Enable interrupt on Analog Watchdog */

ADC1->CFGR1 |= ADC_CFGR1_CONT
| (17 << 26) | ADC_CFGR1_AWDEN | ADC_CFGR1_AWDSDL; /* (1) */
ADC1->TR = (vrefint_high << 16) + vrefint_low; /* (2) */
ADC1->IER = ADC_IER_AWDIE; /* (3) */
A.7.15  Temperature configuration

/* (1) Select CHSEL16 for temperature sensor */
/* (2) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater than
17.1us */
/* (3) Wake-up the Temperature sensor (only for VBAT, Temp sensor and
VRefInt) */
ADC1->CHSELR = ADC_CHSELR_CHSEL16; /* (1) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (2) */
ADC->CCR |= ADC_CCR_TSEN; /* (3) */

A.7.16  Temperature computation

/* Temperature sensor calibration value address */
#define TEMP30_CAL_ADDR ((uint16_t*) ((uint32_t) 0x1FFFF7B8))
#define VDD_CALIB ((uint32_t) (3300))
#define VDD_APPLI ((uint32_t) (3000))
#define AVG_SLOPE ((uint32_t) (5336)) //AVG_SLOPE in ADC conversion step
(@3.3V)/°C multiplied by 1000 for precision on the division

int32_t temperature; /* will contain the temperature in degrees Celsius */
temperature = (((uint32_t) ADC1->DR * VDD_APPLI / VDD_CALIB)) * 1000
- (((uint32_t) *TEMP30_CAL_ADDR;

temperature = (temperature / AVG_SLOPE) + 30;
A.8 Timers

A.8.1 Upcounter on TI2 rising edge

/* (1) Enable the peripheral clock of Timer 1 */
/* (2) Enable the peripheral clock of GPIOA */
/* (3) Select Alternate function mode (10) on GPIOA pin 9 */
/* (4) Select TIM1_CH2 on PA9 by enabling AF2 for pin 9 in GPIOA AFRH register */
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (2) */
GPIOA->MODER = (GPIOA->MODER & ~GPIO_MODER_MODER9) |
               (GPIO_MODER_MODER9_1); /* (3) */
GPIOA->AFR[1] |= 0x2 << ((9-8)*4); /* (4) */
/* (1) Configure channel 2 to detect rising edges on the TI2 input by
writing CC2S = '01', and 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).*/
/* (2) Select rising edge polarity by writing CC2P=0 in the TIMx_CCR
register (reset value). */
/* (3) Configure the timer in external clock mode 1 by writing SMS=111
Select TI2 as the trigger input source by writing TS=110
in the TIMx_SMCR register. */
/* (4) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_IC2F_0 | TIM_CCMR1_IC2F_1 |
              TIM_CCMR1_CC2S_0; /* (1) */
TIMx->CCER &= (uint16_t)(~TIM_CCER_CC2P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
A.8.2 Up counter on each 2 ETR rising edges

/* (1) Enable the peripheral clock of Timer 1 */
/* (2) Enable the peripheral clock of GPIOA */
/* (3) Select Alternate function mode (10) on GPIOA pin 12 */
/* (4) Select TIM1_ETR on PA12 by enabling AF2 for pin 12 in GPIOA AFRH register */
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (2) */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODER12))
| (GPIO_MODER_MODER12_1); /* (3) */
GPIOA->AFR[1] |= 0x2 << ((12-8)*4); /* (4) */
/* (1) As no filter is needed in this example, write ETF[3:0]=0000 in the TIMx_SMCR register. Keep the reset value. 
Set the prescaler by writing ETPS[1:0]=01 in the TIMx_SMCR register. 
Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx_SMCR register. Keep the reset value. 
Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR register. */
/* (2) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->SMCR |= TIM_SMCR_ETPS_0 | TIM_SMCR_ECE; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */

A.8.3 Input capture configuration

/* (1) Select the active input TI1 (CC1S = 01), 
program the input filter for 8 clock cycles (IC1F = 0011), 
select the rising edge on CC1 (CC1P = 0, reset value) 
and prescaler at each valid transition (IC1PS = 00, reset value) */
/* (2) Enable capture by setting CC1E */
/* (3) Enable interrupt on Capture/Compare */
/* (4) Enable counter */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0
| TIM_CCMR1_IC1F_0 | TIM_CCMR1_IC1F_1; /* (1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (2) */
TIMx->DIER |= TIM_DIER_CC1IE; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
A.8.4 Input capture data management

This code must be inserted in the Timer interrupt subroutine.

```c
if ((TIMx->SR & TIM_SR_CC1IF) != 0)
{
    if ((TIMx->SR & TIM_SR_CC1OF) != 0) /* Check the overflow */
    {
        /* Overflow error management */
        gap = 0; /* Reinitialize the laps computing */
        TIMx->SR &= ~(TIM_SR_CC1OF | TIM_SR_CC1IF); /* Clear the flags */
        return;
    }
    if (gap == 0) /* Test if it is the first rising edge */
    {
        counter0 = TIMx->CCR1; /* Read the capture counter which clears the CClIF */
        gap = 1; /* Indicate that the first rising edge has yet been detected */
    }
    else
    {
        counter1 = TIMx->CCR1; /* Read the capture counter which clears the CClIF */
        if (counter1 > counter0) /* Check capture counter overflow */
        {
            Counter = counter1 - counter0;
        }
        else
        {
            Counter = counter1 + 0xFFFF - counter0 + 1;
        }
        counter0 = counter1;
    }
}
else
{
    /* Unexpected Interrupt */
    /* Manage an error for robust application */
}
```

Note: This code manages only a single counter overflow. To manage many counter overflows the update interrupt must be enabled (UIE = 1) and properly managed.
A.8.5 PWM input configuration

/* (1) Select the active input TI1 for TIMx_CCR1 (CC1S = 01),
   select the active input TI1 for TIMx_CCR2 (CC2S = 10) */
/* (2) Select TI1FP1 as valid trigger input (TS = 101)
   configure the slave mode in reset mode (SMS = 100) */
/* (3) Enable capture by setting CC1E and CC2E
   select the rising edge on CC1 and CC1N (CC1P = 0 and CC1NP = 0, reset value),
   select the falling edge on CC2 (CC2P = 1). */
/* (4) Enable interrupt on Capture/Compare 1 */
/* (5) Enable counter */

TIMx->CCMR1 |= TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC2S_1; /* (1) */
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_TS_0
| TIM_SMCR_SMS_2; /* (2) */
TIMx->CCER |= TIM_CCER_CC1E | TIM_CCER_CC2E | TIM_CCER_CC2P; /* (3) */
TIMx->DIER |= TIM_DIER_CC1IE; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */

A.8.6 PWM input with DMA configuration

/* (1) Enable the peripheral clock on DMA */
/* (2) Configure the peripheral data register address for DMA channel x */
/* (3) Configure the memory address for DMA channel x */
/* (4) Configure the number of DMA transfers to be performed
   on DMA channel x */
/* (5) Configure no increment (reset value), size (16-bits), interrupts,
   transfer from peripheral to memory and circular mode
   for DMA channel x */
/* (6) Enable DMA Channel x */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
DMA1_Channel2->CPAR = (uint32_t) (&(TIM1->CCR1)); /* (2) */
DMA1_Channel2->CMAR = (uint32_t) (&Period); /* (3) */
DMA1_Channel2->CNDTR = 1; /* (4) */
DMA1_Channel2->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
| DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel2->CCR |= DMA_CCR_EN; /* (6) */
/* repeat (2) to (6) for channel 3 */
DMA1_Channel3->CPAR = (uint32_t) (&(TIM1->CCR2)); /* (2) */
DMA1_Channel3->CMAR = (uint32_t) (&DutyCycle); /* (3) */
DMA1_Channel3->CNDTR = 1; /* (4) */
DMA1_Channel3->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
| DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel3->CCR |= DMA_CCR_EN; /* (6) */
/* Configure NVIC for DMA */
/* (7) Enable Interrupt on DMA Channels x */
/* (8) Set priority for DMA Channels x */
NVIC_EnableIRQ(DMA1_Channel2_3_IRQn); /* (7) */
NVIC_SetPriority(DMA1_Channel2_3_IRQn, 3); /* (8) */
A.8.7 Output compare configuration

/* (1) Set prescaler to 3, so APBCLK/4 i.e 12MHz */
/* (2) Set ARR = 12000 -1 */
/* (3) Set CCRx = ARR, as timer clock is 12MHz, an event occurs each 1 ms */
/* (4) Select toggle mode on OC1 (OC1M = 011),
   disable preload register on OC1 (OC1PE = 0, reset value) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Enable counter */
TIMx->PSC |= 3; /* (1) */
TIMx->ARR = 12000 - 1; /* (2) */
TIMx->CCR1 = 12000 - 1; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */

A.8.8 Edge-aligned PWM configuration example

/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz the period is 9 us */
/* (3) Set CCRx = 4, the signal will be high during 4 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
   enable preload register on OC1 (OC1PE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Enable counter (CEN = 1)
   select edge aligned mode (CMS = 00, reset value)
   select direction as upcounter (DIR = 0, reset value) */
/* (8) Force update generation (UG = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 4; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
   | TIM_CCMR1_OC1PE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
TIMx->EGR |= TIM_EGR_UG; /* (8) */
A.8.9 Center-aligned PWM configuration example

/* (1) Set prescaler to 47, so APBCLK/48 i.e. 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz and center-aligned counting, 
the period is 16 us */
/* (3) Set CCRx = 7, the signal will be high during 14 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110), 
enable preload register on OC1 (OC1PE = 1, reset value) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value), 
enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Enable counter (CEN = 1) 
select center-aligned mode 1 (CMS = 01) */
/* (8) Force update generation (UG = 1) */

TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CC1 = 7; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
 | TIM_CCMR1_OC1PE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CMS_0 | TIM_CR1_CEN; /* (7) */
TIMx->EGR |= TIM_EGR_UG; /* (8) */
A.8.10 ETR configuration to clear OCxREF

/* This code is similar to the edge-aligned PWM configuration but it enables
the clearing on OC1 for ETR clearing (OC1CE = 1) in CCMR1 (5) and ETR is
configured in SMCR (7). */
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz the period is 9 us */
/* (3) Set CCRx = 4, , the signal will be high during 4 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
enable preload register on OC1 (OC1PE = 1),
enable clearing on OC1 for ETR clearing (OC1CE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Select ETR as OCREF clear source (OCCS = 1),
select External Trigger Prescaler off (ETPS = 00, reset value),
disable external clock mode 2 (ECE = 0, reset value),
select active at high level (ETP = 0, reset value) */
/* (8) Enable counter (CEN = 1),
select edge aligned mode (CMS = 00, reset value),
select direction as upcounter (DIR = 0, reset value) */
/* (9) Force update generation (UG = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 4; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1PE
| TIM_CCMR1_OC1CE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->SMCR |= TIM_SMCR_OCCS; /* (7) */
TIMx->CR1 |= TIM_CR1_CEN; /* (8) */
TIMx->EGR |= TIM_EGR_UG; /* (9) */

A.8.11 Encoder interface

/* (1) Configure TI1FP1 on TI1 (CC1S = 01),
configure TI1FP2 on TI2 (CC2S = 01) */
/* (2) Configure TI1FP1 and TI1FP2 non inverted (CC1P = CC2P = 0, reset
value) */
/* (3) Configure both inputs are active on both rising and falling edges
(SMS = 011) */
/* (4) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC2S_0; /* (1)*/
TIMx->CCER &= (uint16_t)(~(TIM_CCER_CC21 | TIM_CCER_CC2P)); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_0 | TIM_SMCR_SMS_1; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
A.8.12  Reset mode

/* (1) Configure channel 1 to detect rising edges on the TI1 input by writing CC1S = ‘01’, and configure the input filter duration by writing the IC1F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC1F=0000). */
/* (2) Select rising edge polarity by writing CC1P=0 in the TIMx_CCER register Not necessary as it keeps the reset value. */
/* (3) Configure the timer in reset mode by writing SMS=100 Select TI1 as the trigger input source by writing TS=101 in the TIMx_SMCR register. */
/* (4) Set prescaler to 48000-1 in order to get an increment each 1ms */
/* (5) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */

TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (1) */
TIMx->CCER &= (uint16_t)(~TIM_CCER_CC1P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (3) */
TIM1->PSC = 47999; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */

A.8.13  Gated mode

/* (1) Configure channel 1 to detect low level on the TI1 input by writing CC1S = ‘01’, and configure the input filter duration by writing the IC1F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC1F=0000). */
/* (2) Select polarity by writing CC1P=1 in the TIMx_CCER register */
/* (3) Configure the timer in gated mode by writing SMS=101 Select TI1 as the trigger input source by writing TS=101 in the TIMx_SMCR register. */
/* (4) Set prescaler to 12000-1 in order to get an increment each 250us */
/* (5) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */

TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (1) */
TIMx->CCER |= TIM_CCER_CC1P; /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0 | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (3) */
TIMx->PSC = 11999; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */
A.8.14  Trigger mode

/* (1) Configure channel 2 to detect rising edge on the TI2 input
   by writing CC2S = '01',
   and configure the input filter duration by writing the IC1F[3:0]
   bits in the TIMx_CCMR1 register (if no filter is needed,
   keep IC1F=0000). */
/* (2) Select polarity by writing CC2P=0 (reset value) in the TIMx_CCER
   register */
/* (3) Configure the timer in trigger mode by writing SMS=110
   Select TI2 as the trigger input source by writing TS=110
   in the TIMx_SMCR register. */
/* (4) Set prescaler to 12000-1 in order to get an increment each 250us */
TIMx->CCMR1 |= TIM_CCMR1_CC2S_0; /* (1) */
TIMx->CCER &= ~TIM_CCER_CC2P; /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
       | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3) */
TIM1->PSC = 11999; /* (4) */

A.8.15  External clock mode 2 + trigger mode

/* (1) Configure no input filter (ETF=0000, reset value)
   configure prescaler disabled (ETPS = 0, reset value)
   select detection on rising edge on ETR (ETP = 0, reset value)
   enable external clock mode 2 (ECE = 1) */
/* (2) Configure no input filter (IC1F=0000, reset value)
   select input capture source on TI1 (CC1S = 01) */
/* (3) Select polarity by writing CC1P=0 (reset value) in the TIMx_CCER
   register */
/* (4) Configure the timer in trigger mode by writing SMS=110
   Select TI1 as the trigger input source by writing TS=101
   in the TIMx_SMCR register. */
TIMx->SMCR |= TIM_SMCR_ECE; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (2) */
TIMx->CCER &= ~TIM_CCER_CC1P; /* (3) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
       | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (4) */
/* Use TI2FP2 as trigger 1 */
/* (1) Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx_CCMR1 register */
/* (2) TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP=0
   in the TIMx_CCR register (keep the reset value) */
/* (3) 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) */
TIMx->CCMR1 |= TIM_CCMR1_CC2S_0; /* (1) */
TIMx->CCER &= ~(TIM_CCER_CC2P | TIM_CCER_CC2NP); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
       | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3) */
A.8.16 One-Pulse mode

/* The OPM waveform is defined by writing the compare registers */
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 7, as timer clock is 1MHz the period is 8 us */
/* (3) Set CCRx = 5, the burst will be delayed for 5 us (must be > 0) */
/* (4) Select PWM mode 2 on OC1 (OC1M = 111), 
   enable preload register on OC1 (OC1PE = 1, reset value)
   enable fast enable (no delay) if PULSE WITHOUT_DELAY is set */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Write '1 in the OPM bit in the TIMx_CR1 register to stop the counter 
       at the next update event (OPM = 1), 
   enable auto-reload register (ARPE = 1) */

TIMx->PSC = 47; /* (1) */
TIMx->ARR = 7; /* (2) */
TIMx->CCR1 = 5; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_0
            | TIM_CCMR1_OC1PE
#if PULSE WITHOUT_DELAY > 0
    | TIM_CCMR1_OC1FE
#endif
        ; /* (4) */
TIMx->CCER |= TIM_CCRER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_OPM | TIM_CR1_ARPE; /* (7) */

A.8.17 Timer prescaling another timer

/* TIMy is slave of TIMx */
/* (1) Select Update Event as Trigger output (TRGO) by writing MMS = 010 
in TIMx_CR2. */
/* (2) Configure TIMy in slave mode using ITI1 as internal trigger 
by writing TS = 000 in TIMy_SMCR (reset value) 
Configure TIMy in external clock mode 1, by writing SMS=111 in the 
TIMy_SMCR register. */
/* (3) Set TIMx prescaler to 47999 in order to get an increment each 1ms */
/* (4) Set TIMx Autoreload to 999 in order to get an overflow (so an UEV) 
each second */
/* (5) Set TIMx Autoreload to 24*3600-1 in order to get an overflow each 24-
  hour */
/* (6) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
/* (7) Enable the counter by writing CEN=1 in the TIMy_CR1 register. */
TIMx->CR2 |= TIM_CR2_MMS_1; /* (1) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1 | TIM_SMCR_SMS_0; /* (2) */
TIMx->PSC = 47999; /* (3) */
TIMx->ARR = 999; /* (4) */
TIMy->ARR = (24 * 3600) - 1; /* (5) */
TIMx->CR1 |= TIM_CR1_CEN; /* (6) */
TIMy->CR1 |= TIM_CR1_CEN; /* (7) */
A.8.18 Timer enabling another timer

/* TIMy is slave of TIMx */
/* (1) Configure Timer x master mode to send its Output Compare 1 Reference
   (OC1REF) signal as trigger output
   (MMS=100 in the TIM1_CR2 register). */
/* (2) Configure the Timer x OC1REF waveform (TIM1_CCMR1 register)
   Channel 1 is in PWM mode 1 when the counter is less than the
   capture/compare register (write OC1M = 110) */
/* (3) Configure TIMy in slave mode using ITR1 as internal trigger
   by writing TS = 000 in TIMy_SMCR (reset value)
   Configure TIMy in gated mode, by writing SMS=101 in the
   TIMy_SMCR register. */
/* (4) Set TIMx prescaler to 2 */
/* (5) Set TIMy prescaler to 2 */
/* (6) Set TIMx Autoreload to 999 in order to get an overflow (so an UEV)
   each 100ms */
/* (7) Set capture compare register to a value between 0 and 999 */
TIMx->CR2 |= TIM_CR2_MMS_2; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0; /* (3) */
TIMx->PSC = 2; /* (4) */
TIMy->PSC = 2; /* (5) */
TIMx->ARR = 999; /* (6) */
TIMy->CCR1 = 700; /* (7) */
/* Configure the slave timer to generate toggling on each count */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy Autoreload to 1 */
/* (3) Set capture compare register to 1 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->ARR = 1; /* (2) */
TIMy->CCR1 = 1; /* (3) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* (1) Enable the slave counter first by writing CEN=1
   in the TIMy_CR1 register. */
/* (2) Enable the master counter by writing CEN=1
   in the TIMx_CR1 register. */
TIMy->CR1 |= TIM_CR1_CEN; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */
A.8.19 Master and slave synchronization

/* (1) Configure Timer x master mode to send its enable signal as trigger output (MMS=001 in the TIM1_CR2 register). */
/* (2) Configure the Timer x Channel 1 waveform (TIM1_CCMR1 register) is in PWM mode 1 (write OC1M = 110). */
/* (3) Configure TIMy in slave mode using ITR1 as internal trigger by writing TS = 000 in TIMy_SMCR (reset value)
   Configure TIMy in gated mode, by writing SMS=101 in the TIMy_SMCR register. */
/* (4) Set TIMx prescaler to 2 */
/* (5) Set TIMy prescaler to 2 */
/* (6) Set TIMx Autoreload to 99 in order to get an overflow (so an UEV)
each 10ms */
/* (7) Set capture compare register to a value between 0 and 99 */
TIMx->CR2 |= TIM_CR2_MMS_0; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0; /* (3) */
TIMx->PSC = 2; /* (4) */
TIMy->PSC = 2; /* (5) */
TIMx->ARR = 99; /* (6) */
TIMy->CCR1 = 25; /* (7) */

/* Configure the slave timer Channel 1 as PWM as Timer to show synchronicity */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy Autoreload to 99 */
/* (3) Set capture compare register to 25 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->ARR = 99; /* (2) */
TIMy->CCR1 = 25; /* (3) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* (1) Reset Timer x by writing '1 in UG bit (TIMx_EGR register) */
/* (2) Reset Timer y by writing '1 in UG bit (TIMy_EGR register) */
TIMx->EGR |= TIM_EGR_UG; /* (1) */
TIMy->EGR |= TIM_EGR_UG; /* (2) */
/* (1) Enable the slave counter first by writing CEN=1 in the TIMy_CR1 register.
   TIMy will start synchronously with the master timer */
/* (2) Start the master counter by writing CEN=1 in the TIMx_CR1 register. */
TIMy->CR1 |= TIM_CR1_CEN; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */
A.8.20  Two timers synchronized by an external trigger

/* (1) Configure TIMx master mode to send its enable signal as trigger output (MMS=001 in the TIM1_CR2 register). */
/* (2) Configure TIMx in slave mode to get the input trigger from TI1 by writing TS = 100 in TIMx_SMCR
 Configure TIMx in trigger mode, by writing SMS=110 in the TIMx_SMCR register.
 Configure TIMx in Master/Slave mode by writing MSM = 1 in TIMx_SMCR */
/* (3) Configure TIMy in slave mode to get the input trigger from Timer1 by writing TS = 100 in TIMy_SMCR (reset value)
 Configure TIMy in trigger mode, by writing SMS=110 in the TIMy_SMCR register. */
/* (4) Reset Timer x counter by writing '1 in UG bit (TIMx_EGR register) */
/* (5) Reset Timer y counter by writing '1 in UG bit (TIMy_EGR register) */
TIMx->CR2 |= TIM_CR2_MMS_0; /* (1) */
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1 |
TIM_SMCR_MSM; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1; /* (3) */
TIMy->EGR |= TIM_EGR_UG; /* (4) */
TIMy->EGR |= TIM_EGR_UG; /* (5) */
/* Configure the Timer Channel 2 as PWM */
/* (1) Configure the Timer x Channel 2 waveform (TIM1_CCMR1 register)
is in PWM mode 1 (write OC2M = 110) */
/* (2) Set TIMx prescaler to 2 */
/* (3) Set TIMx Autoreload to 99 in order to get an overflow (so an UEV) each 10ms */
/* (4) Set capture compare register to a value between 0 and 99 */
TIMx->CCMR1 |= TIM_CCMR1_OC2M_2 | TIM_CCMR1_OC2M_1; /* (1) */
TIMx->PSC = 2; /* (2) */
TIMx->ARR = 99; /* (3) */
TIMx->CCR2 = 25; /* (4) */
/* Configure the slave timer Channel 1 as PWM as Timer to show synchronicity */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy prescaler to 2 */
/* (3) Set TIMx Autoreload to 99 */
/* (4) Set capture compare register to 25 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->PSC = 2; /* (2) */
TIMy->ARR = 99; /* (3) */
TIMy->CCR1 = 25; /* (4) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value), enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC2E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value), enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
A.8.21 DMA burst feature

/* In this example TIMx has been previously configured
in PWM center-aligned */
/* Configure DMA Burst Feature */
/* Configure the corresponding DMA channel */
/* (1) Set DMA channel peripheral address is the DMAR register address */
/* (2) Set DMA channel memory address is the address of the buffer
in the RAM containing the data to be transferred by DMA
into CCRx registers */
/* (3) Set the number of data transfer to sizeof(Duty_Cycle_Table) */
/* (4) Configure DMA transfer in CCR register,
   enable the circular mode by setting CIRC bit (optional),
   set memory size to 16_bits MSIZE = 01,
   set peripheral size to 32_bits PSIZE = 10,
   enable memory increment mode by setting MINC,
   set data transfer direction read from memory by setting DIR. */
/* (5) Configure TIMx_DCR register with DBL = 3 transfers
   and DBA = (@TIMx->CCR2 - @TIMx->CR1) >> 2 = 0xE */
/* (6) Enable the TIMx update DMA request by setting UDE bit in DIER
     register */
/* (7) Enable TIMx */
/* (8) Enable DMA channel */
DMA1_Channel2->CPAR = (uint32_t)&(TIMx->DMAR); /* (1) */
DMA1_Channel2->CMAR = (uint32_t)(Duty_Cycle_Table); /* (2) */
DMA1_Channel2->CNDTR = 10*3; /* (3) */
DMA1_Channel2->CCR |= DMA_CCR_CIRC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_1 |
                     | DMA_CCR_MINC | DMA_CCR_DIR; /* (4) */
TIMx->DCR = (3 << 8)
            + (((uint32_t)&TIMx->CCR2)
               - ((uint32_t)&TIMx->CR1)) >> 2; /* (5) */
TIMx->DIER |= TIM_DIER_UDE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
DMA1_Channel2->CCR |= DMA_CCR_EN; /* (8) */
A.9 IRTIM code examples

A.9.1 TIM16 and TIM17 configuration

/* The following configuration is for RC5 standard */
/* TIM16 is used for the enveloppe while TIM17 is used for the carrier */
#define TIM_ENV TIM16
#define TIM_CAR TIM17
/* (1) Enable the peripheral clocks of Timer 16 and 17 and SYSCFG */
/* (2) Enable the peripheral clock of GPIOB */
/* (3) Select alternate function mode on GPIOB pin 9 */
/* (4) Select AF0 on PB9 in APFIH for IR_OUT (reset value) */
/* (5) Enable the high sink driver capability by setting I2C_PB9_FM+ bit in SYSCFG_CFG1 */
RCC->APB2ENR |= RCC_APB2ENR_TIM16EN | RCC_APB2ENR_TIM17EN
| RCC_APB2ENR_SYSCFGCOMPEN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOBEN; /* (2) */
GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER9)
| GPIO_MODER_MODER9_1; /* (3) */
GPIOB->AFR[1] &= 0x0F << ((9 - 8) * 4)); /* (4) */
SYSCFG->CFG1 = SYSCFG_CFG1_I2C_FMP_PB9; /* (5) */
/* Configure TIM_CAR as carrier signal */
/* (1) Set prescaler to 1, so APBCLK i.e 48MHz */
/* (2) Set ARR = 1333, as timer clock is 48MHz the frequency is 36kHz */
/* (3) Set CCRx = 1333/4, , the signal will have a 25% duty cycle */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
enable preload register on OC1 (OC1PE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
TIM_CAR->PSC = v; /* (1) */
TIM_CAR->ARR = 1333; /* (2) */
TIM_CAR->CCMR1 = (uint16_t)(1333 / 4); /* (3) */
TIM_CAR->CCER | TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
| TIM_CCMR1_OC1PE; /* (4) */
TIM_CAR->BDTR | TIM_CCRER_CC1E; /* (5) */
/* Configure TIM_ENV is the modulation enveloppe */
/* (1) Set prescaler to 1, so APBCLK i.e 48MHz */
/* (2) Set ARR = 42627, as timer clock is 48MHz the period is 888 us */
/* (3) Select Forced inactive on OC1 (OC1M = 100) */
/* (4) Select active high polarity on OC1 (CC1P = 0, reset value),
enable the output on OC1 (CC1E = 1) */
/* (5) Enable output (MOE = 1) */
/* (6) Enable Update interrupt (UIE = 1) */
TIM_ENV->PSC = 0; /* (1) */
TIM_ENV->ARR = 42627; /* (2) */
TIM_ENV->CCMR1 = TIM_CCMR1_OC1M_2; /* (3) */
TIM_ENV->CCER | TIM_CCRER_CC1E; /* (4) */
TIM_ENV->BDTR | TIM_BDTR_MOE; /* (5) */
TIM_ENV->DIER | TIM_DIER_UIE; /* (6) */
/* Enable and reset TIM_CAR only */
/* (1) Enable counter (CEN = 1),
select edge aligned mode (CMS = 00, reset value),
select direction as upcounter (DIR = 0, reset value) */
A.9.2 IRQHandler for IRTIM

/**
 * Description: This function handles TIM_16 interrupt request.
 *              This interrupt subroutine computes the laps between 2
 *              rising edges on T1IC.
 *              This laps is stored in the "Counter" variable.
 */
void TIM16_IRQHandler(void)
{
    uint8_t bit_msg = 0;

    if ((SendOperationReady == 1)
        && (BitsSentCounter < (RC5_GlobalFrameLength * 2)))
    {
        if (BitsSentCounter < 32)
        {
            SendOperationCompleted = 0x00;
            bit_msg = (uint8_t)((ManchesterCodedMsg >> BitsSentCounter) & 1);

            if (bit_msg== 1)
            {
                /* Force active level - OC1REF is forced high */
                TIM_ENV->CCMR1 |= TIM_CCMR1_OC1M_0;
            }
            else
            {
                /* Force inactive level - OC1REF is forced low */
                TIM_ENV->CCMR1 &= (uint16_t)(~TIM_CCMR1_OC1M_0);
            }
            BitsSentCounter++;
        }
        else
        {
            SendOperationCompleted = 0x01;
            SendOperationReady = 0;
            BitsSentCounter = 0;
        }
        /* Clear TIM_ENV update interrupt */
        TIM_ENV->SR &= (uint16_t)(~TIM_SR_UIF);
    }
A.10 DBG code examples

A.10.1 DBG read device ID

/* Read MCU Id, 32-bit access */
MCU_id = DBGMCU->IDCODE;

A.10.2 DBG debug in Low-power mode

/* To be able to debug in stop mode */
DBGMCU->CR |= DBGMCU_CR_DBG_STOP;

A.11 I2C code examples

A.11.1 I2C configured in master mode to receive

/* (1) Timing register value is computed with the AN4235 xls file,
fast Mode 840kHz with I2CCLK = 48MHz, rise time = 140ns,
fall time = 40ns */
/* (2) Periph enable, receive interrupt enable */
/* (3) Slave address = 0x5A, read transfer, 1 byte to receive, autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE | I2C_CR1_RXIE; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (1<<16) | I2C_CR2_RD_WRN
   | (I2C1_OWN_ADDRESS << 1); /* (3) */

A.11.2 I2C configured in master mode to transmit

/* (1) Timing register value is computed with the AN4235 xls file,
fast Mode 840kHz with I2CCLK = 48MHz, rise time = 140ns,
fall time = 40ns */
/* (2) Periph enable */
/* (3) Slave address = 0x5A, write transfer, 1 byte to transmit, autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (1 << 16) | (I2C1_OWN_ADDRESS << 1); /* (3) */
A.11.3 I2C configured in slave mode

/* (1) Timing register value is computed with the AN4235 xls file,
fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
fall time = 40ns */
/* (2) Periph enable, address match interrupt enable */
/* (3) 7-bit address = 0x5A */
/* (4) Enable own address 1 */
I2C1->TIMINGR = (uint32_t)0x00B00000; /* (1) */
I2C1->CR1 = I2C_CR1_PE | I2C_CR1_ADDRIE; /* (2) */
I2C1->OAR1 |= (uint32_t)(I2C1_OWN_ADDRESS << 1); /* (3) */
I2C1->OAR1 |= I2C_OAR1_OA1EN; /* (4) */

A.11.4 I2C master transmitter

/* Check Tx empty */
if ((I2C2->ISR & I2C_ISR_TXE) == I2C_ISR_TXE)
{
    I2C2->TXDR = I2C_BYTE_TO_SEND; /* Byte to send */
    I2C2->CR2 |= I2C_CR2_START; /* Go */
}

A.11.5 I2C master receiver

if ((I2C2->ISR & I2C_ISR_RXNE) == I2C_ISR_RXNE)
{
    /* Read receive register, will clear RXNE flag */
    if (I2C2->RXDR == I2C_BYTE_TO_SEND)
    {
        /* Process */
    }
}
A.11.6 I2C slave transmitter

```c
uint32_t I2C_InterruptStatus = I2C1->ISR; /* Get interrupt status */
/* Check address match */
if (((I2C_InterruptStatus & I2C_ISR_ADDR) == I2C_ISR_ADDR)) {
   I2C1->ICR |= I2C_ICR_ADDRCF; /* Clear address match flag */
   /* Check if transfer direction is read (slave transmitter) */
   if (((I2C1->ISR & I2C_ISR_DIR) == I2C_ISR_DIR)) {
      I2C1->CR1 |= I2C_CR1_TXIE; /* Set transmit IT */
   }
} else if (((I2C_InterruptStatus & I2C_ISR_TXIS) == I2C_ISR_TXIS)) {
   I2C1->CR1 &=~ I2C_CR1_TXIE; /* Disable transmit IT */
   I2C1->TXDR = I2C_BYTE_TO_SEND; /* Byte to send */
}
```

A.11.7 I2C slave receiver

```c
uint32_t I2C_InterruptStatus = I2C1->ISR; /* Get interrupt status */
if (((I2C_InterruptStatus & I2C_ISR_ADDR) == I2C_ISR_ADDR)) {
   I2C1->ICR |= I2C_ICR_ADDRCF; /* Address match event */
} else if (((I2C_InterruptStatus & I2C_ISR_RXNE) == I2C_ISR_RXNE)) {
   /* Read receive register, will clear RXNE flag */
   if (I2C1->RXDR == I2C_BYTE_TO_SEND) {
      /* Process */
   }
}
```

A.11.8 I2C configured in master mode to transmit with DMA

```c
/* (1) Timing register value is computed with the AN4235 xls file, 
   fast Mode 8400kHz with I2CCLK = 48MHz, rise time = 140ns, 
   fall time = 40ns */
/* (2) Periph enable */
/* (3) Slave address = 0x5A, write transfer, 2 bytes to transmit, 
   autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE | I2C_CR1_TXDMAEN; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (SIZE_OF_DATA << 16) 
   | (I2C1_OWN_ADDRESS << 1); /* (3) */
```
A.11.9 I2C configured in slave mode to receive with DMA

/* (1) Timing register value is computed with the AN4235 xls file, fast Mode 8400kHz with I2CCLK = 48MHz, rise time = 140ns, fall time = 40ns */
/* (2) Periph enable, receive DMA enable */
/* (3) 7-bit address = 0x5A */
/* (4) Enable own address 1 */
I2C1->TIMINGR = (uint32_t)0x00B00000; /* (1) */
I2C1->CR1 = I2C_CR1_PE | I2C_CR1_RXDMAEN | I2C_CR1_ADDRIE; /* (2) */
I2C1->OAR1 |= (uint32_t)(I2C1_OWN_ADDRESS << 1); /* (3) */
I2C1->OAR1 |= I2C_OAR1_OA1EN; /* (4) */

A.12 IWDG code examples

A.12.1 IWDG configuration

/* (1) Activate IWDG (not needed if done in option bytes) */
/* (2) Enable write access to IWDG registers */
/* (3) Set prescaler by 8 */
/* (4) Set reload value to have a rollover each 100ms */
/* (5) Check if flags are reset */
/* (6) Refresh counter */
IWDG->KR = IWDG_START; /* (1) */
IWDG->KR = IWDG_WRITE_ACCESS; /* (2) */
IWDG->PR = IWDG_PR_PR_0; /* (3) */
IWDG->RLR = IWDG_RELOAD; /* (4) */
while (IWDG->SR) /* (5) */
{
    /* add time out here for a robust application */
}
IWDG->KR = IWDG_REFRESH; /* (6) */
A.12.2 IWDG configuration with window

/* (1) Activate IWDG (not needed if done in option bytes) */
/* (2) Enable write access to IWDG registers */
/* (3) Set prescaler by 8 */
/* (4) Set reload value to have a rollover each 100ms */
/* (5) Check if flags are reset */
/* (6) Set a 50ms window, this will refresh the IWDG */
IWDG->KR = IWDG_START; /* (1) */
IWDG->KR = IWDG_WRITE_ACCESS; /* (2) */
IWDG->PR = IWDG_PR_PR_0; /* (3) */
IWDG->RLR = IWDG_RELOAD; /* (4) */
while (IWDG->SR) /* (5) */
{
    /* add time out here for a robust application */
}
IWDG->WINR = IWDG_RELOAD >> 1; /* (6) */

A.13 RTC code examples

A.13.1 RTC calendar configuration

/* (1) Write access for RTC registers */
/* (2) Enable init phase */
/* (3) Wait until it is allow to modify RTC register values */
/* (4) set prescaler, 40kHz/128 => 312 Hz, 312Hz/312 => 1Hz */
/* (5) New time in TR */
/* (6) Disable init phase */
/* (7) Disable write access for RTC registers */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->ISR |= RTC_ISR_INIT; /* (2) */
while ((RTC->ISR & RTC_ISR_INITF) != RTC_ISR_INITF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->PRER = 0x007F0137; /* (4) */
RTC->TR = RTC_TR_PM | Time; /* (5) */
RTC->ISR &=~ RTC_ISR_INIT; /* (6) */
RTC->WPR = 0xFE; /* (7) */
RTC->WPR = 0x64; /* (7) */
A.13.2 RTC alarm configuration

/* (1) Write access for RTC registers */
/* (2) Disable alarm A to modify it */
/* (3) Wait until it is allow to modify alarm A value */
/* (4) Modify alarm A mask to have an interrupt each 1Hz */
/* (5) Enable alarm A and alarm A interrupt */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &=- RTC_CR_ALRAE; /* (2) */
while (((RTC->ISR & RTC_ISR_ALRAWF) != RTC_ISR_ALRAWF) /* (3) */)
{ /* add time out here for a robust application */
  RTC->ALRMAR = RTC_ALRMAR_MSK4 | RTC_ALRMAR_MSK3
  | RTC_ALRMAR_MSK2 | RTC_ALRMAR_MSK1; /* (4) */
  RTC->CR = RTC_CR_ALRAIE | RTC_CR_ALRAE; /* (5) */
  RTC->WPR = 0xFE; /* (6) */
  RTC->WPR = 0x64; /* (6) */
}

A.13.3 RTC WUT configuration

/* (1) Write access for RTC registers */
/* (2) Disable wake up timer to modify it */
/* (3) Wait until it is allow to modify wake up reload value */
/* (4) Modify wake up value reload counter to have a wake up each 1Hz */
/* (5) Enable wake up counter and wake up interrupt */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &=- RTC_CR_WUTE; /* (2) */
while (((RTC->ISR & RTC_ISR_WUTWF) != RTC_ISR_WUTWF) /* (3) */)
{ /* add time out here for a robust application */
  RTC->WUTR = 0x9C0; /* (4) */
  RTC->CR = RTC_CR_WUTE | RTC_CR_WUTIE; /* (5) */
  RTC->WPR = 0xFE; /* (6) */
  RTC->WPR = 0x64; /* (6) */
}

A.13.4 RTC read calendar

if((RTC->ISR & RTC_ISR_RSF) == RTC_ISR_RSF)
{
  TimeToCompute = RTC->TR; /* get time */
  DateToCompute = RTC->DR; /* need to read date also */
}
A.13.5 RTC calibration

/* (1) Write access for RTC registers */
/* (2) Enable init phase */
/* (3) Wait until it is allow to modify RTC register values */
/* (4) Set prescaler, 40kHz/125 => 320 Hz, 320Hz/320 => 1Hz */
/* (5) New time in TR */
/* (6) Disable init phase */
/* (7) Wait until it’s allow to modify calibration register */
/* (8) Set calibration to around +20ppm, which is a standard value @25°C */
/* Note: the calibration is relevant when LSE is selected for RTC clock */
/* (9) Disable write access for RTC registers */

RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->ISR |= RTC_ISR_INIT; /* (2) */
while ((RTC->ISR & RTC_ISR_INITF) != RTC_ISR_INITF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->PRER = (124<<16) | 319; /* (4) */
RTC->TR = RTC_TR_PM | Time; /* (5) */
RTC->ISR &=~ RTC_ISR_INIT; /* (6) */
while((RTC->ISR & RTC_ISR_RECALPF) == RTC_ISR_RECALPF) /* (7) */
{
    /* add time out here for a robust application */
}
RTC->CALR = RTC_CALR_CALP | 482; /* (8) */
RTC->WPR = 0xFE; /* (9) */
RTC->WPR = 0x64; /* (9) */

A.13.6 RTC tamper and time stamp configuration

/* Tamper configuration:
- Disable precharge (PU)
- RTCCLK/256 tamper sampling frequency
- Activate time stamp on tamper detection
- input rising edge trigger detection on RTC_TAMP2 (PA0)
- Tamper interrupt enable */

RTC->TAFCR = RTC_TAFCR_TAMPPUDIS | RTC_TAFCR_TAMPFREQ | RTC_TAFCR_TAMPTS
            | RTC_TAFCR_TAMP2E | RTC_TAFCR_TAMPIE;
A.13.7 RTC tamper and time stamp

/* Check tamper and timestamp flag */
if (((RTC-&gt;ISR &amp; (RTC_ISR_TAMP2F)) == (RTC_ISR_TAMP2F))
    &amp; ((RTC-&gt;ISR &amp; (RTC_ISR_TSF)) == (RTC_ISR_TSF)))
{
    RTC-&gt;ISR &amp;= ~RTC_ISR_TAMP2F; /* clear tamper flag */
    EXTI-&gt;PR = EXTI_PR_PR19; /* clear exti line 19 flag */
    TimeToCompute = RTC-&gt;TSTR; /* get tamper time in timestamp register */
    RTC-&gt;ISR &amp;= ~RTC_ISR_TSF; /* clear timestamp flag */
}

A.13.8 RTC clock output

/* (1) Write access for RTC registers */
/* (2) Disable alarm A to modify it */
/* (3) Wait until it is allow to modify alarm A value */
/* (4) Modify alarm A mask to have an interrupt each 1Hz */
/* (5) Enable alarm A and alarm A interrupt, enable calibration output (1Hz) */
/* (6) Disable write access */
RTC-&gt;WPR = 0xCA; /* (1) */
RTC-&gt;WPR = 0x53; /* (1) */
RTC-&gt;CR &amp;=~ RTC_CR_ALRAE; /* (2) */
while (((RTC-&gt;ISR &amp; RTC_ISR_ALRAWF) != RTC_ISR_ALRAWF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC-&gt;ALRMAR = RTC_ALRMAR_MSK4 | RTC_ALRMAR_MSK3
    | RTC_ALRMAR_MSK2 | RTC_ALRMAR_MSK1; /* (4) */
RTC-&gt;CR = RTC_CR_ALRAIE | RTC_CR_ALRAE | RTC_CR_COE
    | RTC_CR_COSEL; /* (5) */
RTC-&gt;WPR = 0xFE; /* (6) */
RTC-&gt;WPR = 0x64; /* (6) */

A.14 SPI code examples

A.14.1 SPI master configuration

/* (1) Master selection, BR: Fpclk/256 (due to C27 on the board, SPI_CLK is
    set to the minimum) CPOL and CPHA at zero (rising first edge) */
/* (2) Slave select output enabled, RXNE IT, 8-bit Rx fifo */
/* (3) Enable SPI1 */
SPI1-&gt;CR1 = SPI_CR1_MSTR | SPI_CR1_BR; /* (1) */
SPI1-&gt;CR2 = SPI_CR2_SSOE | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
    | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (2) */
SPI1-&gt;CR1 |= SPI_CR1_SPE; /* (3) */
A.14.2 SPI slave configuration

/* nSS hard, slave, CPOL and CPHA at zero (rising first edge) */
/* (1) RXNE IT, 8-bit Rx fifo */
/* (2) Enable SPI2 */
SPI2->CR2 = SPI_CR2_RXNEIE | SPI_CR2_FRXTH |
            SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (1) */
SPI2->CR1 |= SPI_CR1_SPE; /* (2) */

A.14.3 SPI full duplex communication

if ((SPI1->SR & SPI_SR_TXE) == SPI_SR_TXE) /* Test Tx empty */
{ /* Will initiate 8-bit transmission if TXE */
   *(__IO uint8_t *)(SPI1->DR) = SPI1_DATA;
}

A.14.4 SPI interrupt

if ((SPI1->SR & SPI_SR_RXNE) == SPI_SR_RXNE)
{
   SPI1_Data = (uint8_t)SPI1->DR; /* receive data, clear flag */
   /* Process */
}

A.14.5 SPI master configuration with DMA

/* (1) Master selection, BR: Fpclk/256 (due to C27 on the board, SPI_CLK is
   set to the minimum)
/* (2) TX and RX with DMA,
   enable slave select output,
   enable RXNE interrupt,
   select 8-bit Rx fifo */
/* (3) Enable SPI1 */
SPI1->CR1 = SPI_CR1_MSTR | SPI_CR1_BR; /* (1) */
SPI1->CR2 = SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN | SPI_CR2_SSOE |
            SPI_CR2_RXNEIE | SPI_CR2_FRXTH |
            SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (2) */
SPI1->CR1 |= SPI_CR1_SPE; /* (3) */
A.14.6 SPI slave configuration with DMA

/* nSS hard, slave, CPOL and CPHA at zero (rising first edge) */
/* (1) Select TX and RX with DMA, 
   enable RXNE interrupt, 
   select 8-bit Rx fifo */
/* (2) Enable SPI2 */
SPI2->CR2 = SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN
| SPI_CR2_RXNEIE | SPI_CR2_FRXTH
| SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (1) */
SPI2->CR1 |= SPI_CR1_SPE; /* (2) */

A.15 USART code examples

A.15.1 USART transmitter configuration

/* (1) Oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_TE | USART_CR1_UE; /* (2) */

A.15.2 USART transmit byte

/* Start USART transmission */
USART1->TDR = stringtosend[send++]; /* Will initiate TC if TXE is set*/

A.15.3 USART transfer complete

if ((USART1->ISR & USART_ISR_TC) == USART_ISR_TC)
{
  if (send == sizeof(stringtosend))
  {
    send=0;
    USART1->ICR |= USART_ICR_TCCF; /* Clear transfer complete flag */
  }
  else
  {
    /* clear transfer complete flag and fill TDR with a new char */
    USART1->TDR = stringtosend[send++];
  }
}
A.15.4 USART receiver configuration

/* (1) oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity, reception mode */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE; /* (2) */

A.15.5 USART receive byte

if ((USART1->ISR & USART_ISR_RXNE) == USART_ISR_RXNE)
{
    chartoreceive = (uint8_t)(USART1->RDR); /* Receive data, clear flag */
}

A.15.6 USART synchronous mode

/* (1) Oversampling by 16, 9600 baud */
/* (2) Synchronous mode
   CPOL and CPHA = 0 -> rising first edge
   Last bit clock pulse
   Most significant bit first in transmit/receive */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity
   Transmission enabled, reception enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR2 = USART_CR2_MSBFIRST | USART_CR2_CLKEN
    | USART_CR2_LBCL; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
    | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission w/o clock */
while (((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
A.15.7 USART DMA

/* (1) Oversampling by 16, 9600 baud */
/* (2) Enable DMA in reception and transmission */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_DMAT | USART_CR3_DMAR; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */

A.15.8 USART hardware flow control

/* (1) oversampling by 16, 9600 baud */
/* (2) RTS and CTS enabled */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_RTSE | USART_CR3_CTSE; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
    | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */

A.16 WWDG code examples

A.16.1 WWDG configuration

/* (1) Set prescaler to have a roll-over each about 5.5ms,
set window value (about 2.25ms) */
/* (2) Refresh WWDG before activate it */
/* (3) Activate WWDG */
WWDG->CFR = 0x60; /* (1) */
WWDG->CR = WWDG_REFRESH; /* (2) */
WWDG->CR |= WWDG_CR_WDGA; /* (3) */
Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST’s customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture (www.psacertified.org) and/or Security Evaluation standard for IoT Platforms (www.trustcb.com). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on www.st.com for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.

- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.

- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.

- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.

- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.
Table 118. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>23-Sep-2013</td>
<td>1</td>
<td>Initial release</td>
</tr>
<tr>
<td>13-Jan-2015</td>
<td>2</td>
<td>Extended the applicability to STM32F070x6/xB and STM32F030x4/x6/x8/xC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Chapter 30: Universal serial bus full-speed device interface (USB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and reviewed the content of most chapters.</td>
</tr>
<tr>
<td>19-May-2015</td>
<td>3</td>
<td>Renamed TIM16 and TIM17 in the table TIMx internal trigger connection in the section General purpose timer (TIM15/16/17).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Table 85: STM32F0x0 USART implementation for STM32F070xB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Section Appendix A: Code examples.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Bit 8 (DBP) description of PWR_CR register in Section 6.4.1: Power control register (PWR_CR).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– TIM1.ARR reset value in Section 16.4.12: TIM1 auto-reload register (TIM1.ARR).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Bit 3 (TE) description of USART_ISR and Bit 21 description of USARTx_ISR in Section 26.8.8: Interrupt and status register (USART_ISR).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Bit SMS description for encoder mode 1 and encoder mode 2 and added the last note in SMS bit description in Section 16.4.3: TIM1 slave mode control register (TIM1_SMCR), Section 17.4.3: TIM2 and TIM3 slave mode control register (TIM2_SMCR and TIM3_SMCR) and Section 19.5.3: TIM15 slave mode control register (TIM15_SMCR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– the description of ETF bits in Section 16.4.3: TIM1 slave mode control register (TIM1_SMCR) and Section 17.4.3: TIM2 and TIM3 slave mode control register (TIM2_SMCR and TIM3_SMCR).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– the description of IC1F[3:0] bits in Section 16.4.7: TIM1 capture/compare mode register 1 (TIM1_CCMR1), Section 17.4.7: TIM2 and TIM3 capture/compare mode register 1 (TIM2_CCMR1 and TIM3_CCMR1) and Section 19.5.7: TIM15 capture/compare mode register 1 (TIM15_CCMR1).</td>
</tr>
</tbody>
</table>
### Table 118. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 24-Apr-2017 | 4        | - Section: Option bytes - note and description
- Feature list
- RCC
- Section: RTC domain control register (RCC_BDCR) - bit field LSEDRV
- GPIO
- Section: GPIO alternate function low register (GPIOx_AFRL) (x = A to D, ) and other AF registers - AFR renamed to AFSEL
- DMA
- Section: DMA interrupt flag clear register (DMA_IFCR) - bit description

**Interrupt**
- Table: Vector table - removed "(combined with EXTI line 28)" from row position 29
- Section: External and internal interrupt/event line mapping - line 23 reserved
- Section: Interrupt mask register (EXTI_IMR) - MRx bits renamed to IMx
- Section: Event mask register (EXTI_EMR) - MRx bits renamed to EMx
- Section: Rising trigger selection register (EXTI_RTSR) - TRx renamed to RTx and FTx; RT31 added
- Section: Software interrupt event register (EXTI_SWIER) - SWIERx renamed to SWIx; SWI31 added
- Section: Pending register (EXTI_PR) - PRx renamed to PIFx; PIF31 added

**ADC**
- Section: ADC voltage regulator (ADVREGEN) - calibration software procedure
- Section: ADC on-off control (ADEN, ADDIS, ADRDY) - modified
- Section: Starting conversions (ADSTART) and corresponding bit description
- Section: Temperature sensor and VREFINT channel block diagram - added paragraphs

**TIM1**
- Section: Using the break function

**TIM3**
- removed redundant table before Table: TIM3 internal trigger connection
- Section: TIM3 auto-reload register (TIM3_ARR) reset value
- corrected Table: TIM3 register map and reset values
Table 118. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-Apr-2017</td>
<td>4</td>
<td><strong>TIM6/TIM7</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: TIM6/TIM7 auto-reload register (TIMx_ARR) reset value</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TIM15/16/17</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: Using the break function - break function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– corrected Section: TIMx auto-reload register (TIMx_ARR)(x = 16 to 17)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>IWDG</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– added Section: Behavior in Stop and Standby modes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: IWDG status register (IWDG_SR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>RTC</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: RTC block diagram - figures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section Programming the wakeup timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: Resetting the RTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: Calibration clock output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: RTC interrupts - EXTI replaced with NVIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: RTC control register (RTC_CR) bits SUB1H and ADD1H; added caution at the end</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: RTC initialization and status register (RTC_ISR) bit WUTWF</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>I2C</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Table: STM32F0x0 I2C implementation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Table: Comparison of analog vs. digital filters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Figure: Slave initialization flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: I2C timings - multiple additions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: I2C master mode - information on STM32CubeMX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: Master communication initialization (address phase) - information on 10-bit addressing mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: Control register 2 (I2C_CR2) - bit START - information on 10-bit addressing mode, and bit fields SADD and ADDRCF</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>USART</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Table: STM32F0x0 USART features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: USART functional description - removed 1.5 stop bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Table: Effect of low-power modes on the USART</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Table: USART interrupt requests</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Figure: USART interrupt mapping diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: USART control register 1 (USART_CR1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: USART control register 2 (USART_CR2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section: USART control register 3 (USART_CR3)</td>
</tr>
<tr>
<td>Date</td>
<td>Revision</td>
<td>Changes</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>24-Apr-2017</td>
<td>4</td>
<td>- Section: USART interrupt and status register (USART_ISR)\n- Section “Guard time and prescale register” removed\n- Table: USART register map and reset values\nSection “USART smartcard code example” removed</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>SPI</strong>\n- Table: STM32F0x0 SPI implementation\n- Section: Communications between one master and one slave - NSS pin\n- Section: Half-duplex communication\n- Section Simplex communications\n- Section: Multimaster communication\n- Section: Configuration of SPI - point 1b\n- Section: CRC calculation\n- Section: SPI control register 1 (SPIx_CR1) bit MSTR\n- Section: SPI Rx CRC register (SPIx_RXCRCR) and Section: SPI Tx CRC register (SPIx_TXCRCR) - bits 15:0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Debug</strong>\n- Table: DEV_ID and REV_ID field values added</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Appendix</strong>\n- Section: Alternate function selection sequence code example\n- Section: ADC calibration code example\n- Section: ADC enable sequence code example\n- Section: ADC disable sequence code example\n- Section: Analog watchdog code example\n- Section: Temperature computation code example\n- Section: DMA burst feature code example\n- Sections “USART smartcard mode code example” and “USART IrDA mode code example” removed</td>
</tr>
</tbody>
</table>
Table 118. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>09-May-2023</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cover page</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Introduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Document conventions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 1.1: General information added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 1.2: List of abbreviations for registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>System and memory overview</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 2.3: Embedded SRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 2.5: Boot configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Figure 10: Clock tree (STM32F030x4, STM32F030x6 and STM32F030x8 devices) and Figure 11: Clock tree (STM32F070x6, STM32F070xB and STM32F030xC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 7.4.2: Clock configuration register (RCC_CFGR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSCFG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 9.1.1: SYSCFG configuration register 1 (SYSCFG_CFGR1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 12.3.5: Configuring the ADC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 12.8: Temperature sensor and internal reference voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 17: General-purpose timers (TIM15/16/17)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 21.4: RTC functional description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I2C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– section Enabling and disabling the peripheral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>USART</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Table 86: STM32F0x0 USART features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– section How to derive USARTDIV from USART_BRR register values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– section Transmission using DMA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 23.7.7: USART interrupt and status register (USART_ISR) bits ORE and FE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code examples</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section A.4.2: Alternate function selection sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section A.6.1: NVIC initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section A.7.6: Continuous conversion sequence - software trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section A.7.8: Continuous conversion sequence - hardware trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– section Important security notice added</td>
</tr>
</tbody>
</table>
## Index

### A
- ADC_CCR ........................................ 223
- ADC_CFGR1 ................................... 216
- ADC_CFGR2 ................................... 220
- ADC_CHSELR .................................. 221
- ADC_CR ........................................ 214
- ADC_DR ........................................ 222
- ADC_IER ....................................... 212
- ADC_ISR ....................................... 211
- ADC_SMPR ..................................... 220
- ADC_TR ........................................ 221

### C
- CRC_CR ......................................... 73
- CRC_DR ......................................... 72
- CRC_IDR ....................................... 73
- CRC_INIT ...................................... 74

### D
- DBGMCU_APB1_FZ ............................... 716
- DBGMCU_APB2_FZ ............................... 718
- DBGMCU_CR .................................. 715
- DBGMCU_IDCODE ............................... 708
- DMA_CCRx ..................................... 164
- DMA_CMARx .................................... 168
- DMA_CNDTRx .................................. 166
- DMA_CPARx .................................... 167
- DMA_ICCR ..................................... 163
- DMA_ISR ....................................... 161
- DMA1_CSELR ................................... 168

### E
- EXTI_EMR ....................................... 177
- EXTI_FTSR ..................................... 178
- EXTI_IMR ....................................... 177
- EXTI_PR ........................................ 179
- EXTI_RTSR ..................................... 177
- EXTI_SWIER .................................... 179

### F
- FLASH_ACR ..................................... 59
- FLASH_CR ...................................... 61
- FLASH_KEYR .................................... 59
- FLASH_OPTKEYR ............................... 60
- FLASH_SR ...................................... 61

### G
- GPIOx_AFRH .................................... 139
- GPIOx_AFRL .................................... 138
- GPIOx_BRR ..................................... 139
- GPIOx_BSRR .................................... 137
- GPIOx_IDR ..................................... 136
- GPIOx_LCKR .................................... 137
- GPIOx_MODER .................................. 134
- GPIOx_ODR ..................................... 136
- GPIOx_OSPEDR .................................. 135
- GPIOx_OTYPER .................................. 134
- GPIOx_PUPDR ................................... 135

### I
- I2C_CR1 ......................................... 575
- I2C_CR2 ......................................... 577
- I2C_ICR ......................................... 585
- I2C_ISR ......................................... 583
- I2C_OAR1 ....................................... 579
- I2C_OAR2 ....................................... 580
- I2C_PECR ....................................... 586
- I2C_RXDR ....................................... 587
- I2C_TIMEOUTR ................................. 582
- I2C_TIMINGR ................................... 581
- I2C_TXDR ....................................... 587
- IWDG_KR ........................................ 472
- IWDG_PR ........................................ 473
- IWDG_RLR ....................................... 474
- IWDG_SR ........................................ 475
- IWDG_WINR ..................................... 476

### P
- PWR_CR ......................................... 85
- PWR_CSR ....................................... 86

### R
- RCC_AHBENR .................................... 110
- RCC_AHBRRSTR .................................. 118
- RCC_APB1ENR ................................... 113
- RCC_APB1RSTR .................................. 108
- RCC_APB2ENR ................................... 111
- RCC_APB2RSTR .................................. 107
- RCC_BDCR ....................................... 115
- RCC CFGR ....................................... 101
- RCC_CFGR2 ..................................... 119
<table>
<thead>
<tr>
<th>RM0360</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCC_CFGR3</td>
<td>...................</td>
</tr>
<tr>
<td>RCC_CIR</td>
<td>...................</td>
</tr>
<tr>
<td>RCC_CR</td>
<td>...................</td>
</tr>
<tr>
<td>RCC_CR2</td>
<td>...................</td>
</tr>
<tr>
<td>RCC_CSR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_ALRMAR</td>
<td>........</td>
</tr>
<tr>
<td>RTC_ALRMASSR</td>
<td>........</td>
</tr>
<tr>
<td>RTC_CALR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_CR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_DR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_ISR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_RPR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_SIFTR</td>
<td>..........</td>
</tr>
<tr>
<td>RTC_SSR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_TAFCR</td>
<td>........</td>
</tr>
<tr>
<td>RTC_TR</td>
<td>...................</td>
</tr>
<tr>
<td>RTC_TSDR</td>
<td>..........</td>
</tr>
<tr>
<td>RTC_TSSI</td>
<td>..........</td>
</tr>
<tr>
<td>RTC_TSTR</td>
<td>..........</td>
</tr>
<tr>
<td>RTC_WPR</td>
<td>..........</td>
</tr>
<tr>
<td>RTC_WUTR</td>
<td>..........</td>
</tr>
<tr>
<td>S</td>
<td>.......</td>
</tr>
<tr>
<td>SPlx_CR1</td>
<td>...........</td>
</tr>
<tr>
<td>SPlx_CR2</td>
<td>...........</td>
</tr>
<tr>
<td>SPlx_CRCPR</td>
<td>......</td>
</tr>
<tr>
<td>SPlx_DR</td>
<td>..........</td>
</tr>
<tr>
<td>SPlx_RXCRCR</td>
<td>.......</td>
</tr>
<tr>
<td>SPlx_SR</td>
<td>..........</td>
</tr>
<tr>
<td>SPlx_TXCRCR</td>
<td>.......</td>
</tr>
<tr>
<td>SYSCFG_CFGR1</td>
<td>......</td>
</tr>
<tr>
<td>SYSCFG_CFGR2</td>
<td>......</td>
</tr>
<tr>
<td>SYSCFG_EXTICR1</td>
<td>.....</td>
</tr>
<tr>
<td>SYSCFG_EXTICR2</td>
<td>.....</td>
</tr>
<tr>
<td>SYSCFG_EXTICR3</td>
<td>.....</td>
</tr>
<tr>
<td>SYSCFG_EXTICR4</td>
<td>.....</td>
</tr>
<tr>
<td>T</td>
<td>.......</td>
</tr>
<tr>
<td>TIM1.ARR</td>
<td>...........</td>
</tr>
<tr>
<td>TIM1.BDTR</td>
<td>...........</td>
</tr>
<tr>
<td>TIM1.CCER</td>
<td>..........</td>
</tr>
<tr>
<td>TIM1.CCR1</td>
<td>...........</td>
</tr>
<tr>
<td>TIM1.CCR2</td>
<td>...........</td>
</tr>
<tr>
<td>TIM1.CCR3</td>
<td>...........</td>
</tr>
<tr>
<td>TIM1.CCR4</td>
<td>...........</td>
</tr>
<tr>
<td>TIM1.CNT</td>
<td>..........</td>
</tr>
<tr>
<td>TIM1.CR1</td>
<td>..........</td>
</tr>
<tr>
<td>TIM1.CR2</td>
<td>..........</td>
</tr>
<tr>
<td>TIM1.DCR</td>
<td>..........</td>
</tr>
<tr>
<td>TIM1.DIER</td>
<td>..........</td>
</tr>
<tr>
<td>TIM1.DMAR</td>
<td>..........</td>
</tr>
</tbody>
</table>
TIMx_SR ..................... 349, 377, 453

U
USART_BRR ...................... 628
USART_CR1 ...................... 620
USART_CR2 ...................... 623
USART_CR3 ...................... 626
USART_ICR ...................... 633
USART_ISR ...................... 630
USART_RDR ...................... 634
USART_RQR ...................... 629
USART_RTOR ..................... 628
USART_TDR ...................... 635
USB_ADDRn_RX .................... 701
USB_ADDRn_TX .................... 700
USB_BCDR ....................... 694
USB_BTABLE ..................... 693
USB_CNTR ....................... 687
USB_COUNTn_RX .................. 701
USB_COUNTn_TX .................. 700
USB_DADDR ...................... 692
USB_EPnR ....................... 695
USB_FNR ....................... 692
USB_ISTR ....................... 689
USB_LPMCSR ..................... 693

W
WWDG_CF R ...................... 482
WWDG_CR ....................... 481
WWDG_SR ....................... 482
IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2023 STMicroelectronics – All rights reserved