RM0312
Reference manual

STM8TL5xxx microcontroller family

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM8TL5xxx microcontroller family memory and peripherals.

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

For information on the STM8 SWIM communication protocol and debug module, please refer to the user manual (UM0470).

For more information related to the Flash memory, please refer to the STM8TL5xxx Flash memory programming manual (PM0212) .

For information on the STM8 core, please refer to the STM8 CPU programming manual (PM0044).

Contents

1 Memory and register map 17

1.1 Register description abbreviations 17

2 Central processing unit (CPU) 18

2.1 Introduction 18

2.2 CPU registers 18

2.2.1 Description of CPU registers 18

2.2.2 STM8 CPU register map 22

2.3 Global configuration register (CFG_GCR) 22

2.3.1 Activation level 22

2.3.2 SWIM disable 22

2.3.3 Description of global configuration register (CFG_GCR) 23

2.3.4 Global configuration register map and reset values 23

3 Single wire interface module (SWIM) and debug module (DM) 24

3.1 Introduction 24

3.2 Main features 24

3.3 SWIM modes 24

4 Flash program memory and data EEPROM 25

4.1 Introduction 25

4.2 Glossary 25

4.3 Main Flash memory features 25

4.4 Memory organization 26

4.4.1 Proprietary code area (PCODE) 27

4.4.2 User boot area (UBC) 27

4.4.3 Data EEPROM (DATA) 28

4.4.4 Main program area 28

4.4.5 Option bytes 28

4.5 Memory protection 29

4.5.1 Readout protection 29

4.5.2 Memory access security system (MASS) 29

4.6 Memory programming 31

4.6.1 Byte programming . . . 31
4.6.2 Word programming . . . 31
4.6.3 Block programming . . . 32
4.7 ICP and IAP . . . 33
4.8 Flash registers . . . 36
4.8.1 Flash control register 1 (FLASH_CR1) . . . 36
4.8.2 Flash control register 2 (FLASH_CR2) . . . 37
4.8.3 Flash program memory unprotecting key register (FLASH_PUKR) . . . 37
4.8.4 Data EEPROM unprotection key register (FLASH_DUKR) . . . 38
4.8.5 Flash status register (FLASH_IAPSR) . . . 38
4.8.6 Flash register map and reset values . . . 39

5 Interrupt controller (ITC) . . . 40
5.1 ITC introduction . . . 40
5.2 Interrupt masking and processing flow . . . 40
5.2.1 Servicing pending interrupts . . . 41
5.2.2 Interrupt sources . . . 42
5.3 Interrupts and low power modes . . . 43
5.4 Activation level/low power mode control . . . 44
5.5 Concurrent and nested interrupt management . . . 44
5.5.1 Concurrent interrupt management mode . . . 45
5.5.2 Nested interrupt management mode . . . 45
5.6 External interrupts . . . 47
5.7 Interrupt instructions . . . 47
5.8 Interrupt mapping . . . 48
5.9 ITC and EXTI registers . . . 49
5.9.1 CPU condition code register interrupt bits (CCR) . . . 49
5.9.2 Software priority register x (ITC_SPRx) . . . 50
5.9.3 External interrupt control register 1 (EXTI_CR1) . . . 51
5.9.4 External interrupt control register 2 (EXTI_CR2) . . . 52
5.9.5 External interrupt control register 3 (EXTI_CR3) . . . 53
5.9.6 External interrupt status register 1 (EXTI_SR1) . . . 53
5.9.7 External interrupt status register 2 (EXTI_SR2) . . . 54
5.9.8 External interrupt port select register (EXTI_CONF) . . . 54
5.9.9 ITC and EXTI register map and reset values . . . 55

6 Power supply 56

7 Reset (RST) and voltage detection 57

7.1 “Reset state” and “under reset” definitions 57

7.2 External reset (NRST pin) 57

7.2.1 Asynchronous external reset description 57

7.2.2 Configuring NRST/PA5 pin as general purpose output 58

7.3 Internal reset 58

7.3.1 Power-on reset (POR) 58

7.3.2 Independent watchdog reset 58

7.3.3 SWIM reset 58

7.3.4 Illegal opcode reset 58

7.4 RST registers 59

7.4.1 Reset pin configuration register (RST_CR) 59

7.4.2 Reset status register (RST_SR) 59

7.5 RST register map and reset values 60

8 Clock control (CLK) 61

8.1 Master clock (HSI clock) 62

8.1.1 Peripheral clock gating (PCG) 62

8.2 LSI clock 63

8.3 Configurable clock-output capability (CCO) 63

8.4 CLK registers 63

8.4.1 Clock divider register (CLK_CKDIVR) 63

8.4.2 Peripheral clock gating register 1 (CLK_PCKENR1) 64

8.4.3 Peripheral clock gating register 2 (CLK_PCKENR2) 64

8.4.4 Configurable clock output register (CLK_CCOR) 65

8.4.5 CLK register map and reset values 66

9 Power management 67

9.1 General considerations 67

9.2 Managing the clock for low consumption 68

9.2.1 Slowing the system clocks 68

9.2.2 Peripheral clock gating 68

9.3 Switching peripherals off 68

9.4 Low power modes 69

9.4.1 Wait mode 70
9.4.2 Halt mode 71
9.4.3 Active-halt mode 71
9.5 WFE registers 72
9.5.1 WFE control register 1 (WFE_CR1) 72
9.5.2 WFE control register 2 (WFE_CR2) 73
9.6 WFE register map and reset values 74

10 General purpose I/O ports (GPIO) 75
10.1 Introduction 75
10.2 GPIO main features 75
10.3 Port configuration and usage 76
10.3.1 Input modes 77
10.3.2 Output modes 78
10.4 Reset configuration 78
10.5 Unused I/O pins 78
10.6 Low power modes 78
10.7 Input mode details 79
10.7.1 Alternate function input 79
10.7.2 Interrupt capability 79
10.8 Output mode details 79
10.8.1 Alternate function output 79
10.8.2 Slope control 79
10.9 GPIO registers 80
10.9.1 Port x output data register (Px_ODR) 80
10.9.2 Port x pin input register (Px_IDR) 80
10.9.3 Port x data direction register (Px_DDR) 81
10.9.4 Port x control register 1 (Px_CR1) 81
10.9.5 Port x control register 2 (Px_CR2) 82
10.9.6 GPIO register map and reset values 82

11 System configuration controller (SYSCFG) 83
11.1 Introduction 83
11.2 SYSCFG register 83
11.2.1 SYSCFG remap control register 1 (SYSCFG_RMPCR1) 83
11.2.2 SYSCFG register map and reset values 83

12 Auto-wakeup (AWU) 84
12.1 Introduction 84
12.2 LSI clock measurement 84
12.3 AWU functional description 85
12.3.1 AWU operation 85
12.3.2 Time base selection 86
12.3.3 LSI clock frequency measurement 87
12.4 AWU registers 88
12.4.1 Control/status register (AWU_CSR) 88
12.4.2 Asynchronous prescaler register (AWU_APR) 88
12.4.3 Timebase selection register (AWU_TBR) 89
12.4.4 AWU register map and reset values 89

13 Beeper (BEEP) 90
13.1 Introduction 90
13.2 Beeper functional description 90
13.2.1 Beeper operation 90
13.2.2 Beeper calibration 91
13.3 Beeper registers 91
13.3.1 Beeper control/status register (BEEP_CSR) 91
13.3.2 Beeper register map and reset values 92

14 Window watchdog (WWDG) 93
14.1 Introduction 93
14.2 WWDG main features 93
14.3 WWDG functional description 93
14.4 How to program the watchdog timeout 94
14.5 WWDG low power modes 95
14.6 Hardware watchdog option 96
14.7 WWDG interrupts 96
14.8 WWDG registers 96
14.8.1 Control register (WWDG_CR) 96
14.8.2 Window register (WWDG_WR) 96
14.9 Window watchdog register map and reset values 97

15 Independent watchdog (IWDG) 98
15.1 Introduction 98
15.2 IWDG functional description 98
15.3 IWDG registers 100
15.3.1 Key register (IWDG_KR) 100
15.3.2 Prescaler register (IWDG_PR) 100
15.3.3 Reload register (IWDG_RLR) 101
15.3.4 IWDG register map and reset values 101

16 Timer overview 102
16.1 Timer feature comparison 102
16.2 Glossary of timer signal names 103

17 16-bit general purpose timer (TIM2/TIM3) 105
17.1 Introduction 105
17.2 TIMx main features 105
17.3 TIMx time base unit 107
17.3.1 Reading and writing to the 16-bit counter 108
17.3.2 Write sequence for 16-bit TIMx_ARR register 108
17.3.3 Prescaler 108
17.3.4 Up-counting mode 109
17.3.5 Down-counting mode 111
17.3.6 Center-aligned mode (up/down counting) 113
17.4 TIMx clock/trigger controller 115
17.4.1 Prescaler clock (CK_PSC) 115
17.4.2 Internal clock source 115
17.4.3 External clock source mode 1 116
17.4.4 External clock source mode 2 117
17.4.5 Trigger synchronization 118
17.4.6 Synchronization from other timers 122
17.5 TIMx capture/compare channels 128
17.5.1 Write sequence for 16-bit TIMx_CCRi registers 129
17.5.2 Input stage 129
17.5.3 Input capture mode 130
17.5.4 Output stage 132
17.5.5 Forced output mode 133

17.5.6 Output compare mode 133
17.5.7 PWM mode 134
17.5.8 Using the break function 138
17.5.9 Clearing the OCiREF signal on an external event 139
17.5.10 Encoder interface mode 140
17.6 TIMx interrupts 143
17.6.1 TIMx wait-for-event capability 143
17.7 TIMx registers 144
17.7.1 Control register 1 (TIMx_CR1) 144
17.7.2 Control register 2 (TIMx_CR2) 145
17.7.3 Slave mode control register (TIMx_SMCR) 146
17.7.4 External trigger register (TIMx_ETR) 147
17.7.5 Interrupt enable register (TIMx_IER) 148
17.7.6 Status register 1 (TIMx_SR1) 148
17.7.7 Status register 2 (TIMx_SR2) 149
17.7.8 Event generation register (TIMx_EGR) 150
17.7.9 Capture/compare mode register 1 (TIMx_CCMR1) 151
17.7.10 Capture/compare mode register 2 (TIMx_CCMR2) 153
17.7.11 Capture/compare enable register 1 (TIMx_CCER1) 154
17.7.12 Counter high (TIMx_CNTRH) 155
17.7.13 Counter low (TIMx_CNTRL) 155
17.7.14 Prescaler register (TIMx_PSCR) 155
17.7.15 Auto-reload register high (TIMx_ARRH) 156
17.7.16 Auto-reload register low (TIMx_ARRL) 156
17.7.17 Capture/compare register 1 high (TIMx_CCR1H) 157
17.7.18 Capture/compare register 1 low (TIMx_CCR1L) 157
17.7.19 Capture/compare register 2 high (TIMx_CCR2H) 158
17.7.20 Capture/compare register 2 low (TIMx_CCR2L) 158
17.7.21 Break register (TIMx_BKR) 159
17.7.22 Output idle state register (TIMx_OISR) 161
17.7.23 TIMx register map and reset values 161

18 8-bit basic timer (TIM4) 163
18.1 Introduction 163
18.2 TIM4 main features 163
18.3 TIM4 interrupts 163

18.4 TIM4 clock selection 164
18.5 TIM4 registers 165
18.5.1 Control register 1 (TIM4_CR1) 165
18.5.2 Control register 2 (TIM4_CR2) 166
18.5.3 Slave mode control register (TIM4_SMCR) 167
18.5.4 Interrupt enable register (TIM4_IER) 168
18.5.5 Status register 1 (TIM4_SR1) 168
18.5.6 Event generation register (TIM4_EGR) 169
18.5.7 Counter (TIM4_CNTR) 169
18.5.8 Prescaler register (TIM4_PSCR) 169
18.5.9 Auto-reload register (TIM4_ARR) 170
18.5.10 TIM4 register map and reset values 170

19 Inter-integrated circuit (I^2C) interface 171
19.1 Introduction 171
19.2 I^2C main features 171
19.3 I^2C general description 172
19.4 I^2C functional description 174
19.4.1 I^2C slave mode 174
19.4.2 I^2C master mode 176
19.4.3 Error conditions 182
19.4.4 SDA/SCL line control 183
19.5 I^2C low power modes 184
19.6 I^2C interrupts 184
19.7 I^2C registers 186
19.7.1 Control register 1 (I2C_CR1) 186
19.7.2 Control register 2 (I2C_CR2) 187
19.7.3 Frequency register (I2C_FREQR) 188
19.7.4 Own address register 1 LSB (I2C_OAR1L) 189
19.7.5 Own address register 1 MSB (I2C_OAR1H) 189
19.7.6 Own address register 2 (I2C_OAR2) 190
19.7.7 Data register (I2C_DR) 190
19.7.8 Status register 1 (I2C_SR1) 191
19.7.9 Status register 2 (I2C_SR2) 193
19.7.10 Status register 3 (I2C_SR3) 194
19.7.11 Interrupt register (I2C_ITR) 195

19.7.12 Clock control register low (I2C_CCRL) 196
19.7.13 Clock control register high (I2C_CCRH) 197
19.7.14 TRISE register (I2C_TRISER) 199
19.7.15 I^2C register map and reset values 199

20 Serial peripheral interface (SPI) 201
20.1 Introduction 201
20.2 SPI main features 201
20.3 SPI functional description 202
20.3.1 General description 202
20.3.2 Configuring the SPI in slave mode 206
20.3.3 Configuring the SPI master mode 206
20.3.4 Configuring the SPI for simplex communications 207
20.3.5 Data transmission and reception procedures 207
20.3.6 Status flags 214
20.3.7 Disabling the SPI 215
20.3.8 Error flags 216
20.3.9 SPI low power modes 217
20.3.10 SPI interrupts 218
20.4 SPI registers 219
20.4.1 SPI control register 1 (SPI_CR1) 219
20.4.2 SPI control register 2 (SPI_CR2) 220
20.4.3 SPI interrupt control register (SPI_ICR) 221
20.4.4 SPI status register (SPI_SR) 222
20.4.5 SPI data register (SPI_DR) 223
20.5 SPI register map and reset values 223

21 Universal synchronous/asynchronous receiver transmitter (USART) 224
21.1 USART introduction 224
21.2 USART main features 225
21.3 USART functional description 226
21.3.1 USART character description 228
21.3.2 Transmitter 229
21.3.3 Receiver 231
21.3.4 High precision baud rate generator 235

21.3.5 USART receiver's tolerance to clock deviation 236
21.3.6 Parity control 237
21.3.7 Multi-processor communication 238
21.3.8 USART synchronous communication 239
21.4 USART low power modes 242
21.5 USART interrupts 242
21.6 USART registers 243
21.6.1 Status register (USART_SR) 243
21.6.2 Data register (USART_DR) 244
21.6.3 Baud rate register 1 (USART_BRR1) 245
21.6.4 Baud rate register 2 (USART_BRR2) 245
21.6.5 Control register 1 (USART_CR1) 246
21.6.6 Control register 2 (USART_CR2) 247
21.6.7 Control register 3 (USART_CR3) 248
21.6.8 Control register 4 (USART_CR4) 249
21.6.9 USART register map and reset values 249

22 ProxSense™ (PXS) 250
22.1 Introduction 250
22.2 Main features of the ProxSense peripheral 250
22.3 PXS pins 251
22.4 Functional description 251
22.4.1 General description 251
22.4.2 ProxSense charge transfer sequence 252
22.5 ProxSense operations 253
22.5.1 PXS on/off control 253
22.5.2 PXS initialization 253
22.5.3 PXS acquisition launching 254
22.5.4 PXS acquisition result 255
22.5.5 Stopping a PXS acquisition 255
22.6 Special features 255
22.6.1 Sampling capacitor selection 255
22.6.2 EPCC selection 255
22.6.3 RF detection 255
22.7 Low power modes 256
22.7.1 Behavior in low power modes 256

22.7.2 Reducing power consumption 256
22.8 PXS interrupts 257
22.9 ProxSense registers 258
22.9.1 ProxSense Control Register 1 (PXS_CR1) 258
22.9.2 ProxSense Control Register 2 (PXS_CR2) 259
22.9.3 ProxSense Control Register 3 (PXS_CR3) 260
22.9.4 ProxSense Interrupt and Status Register (PXS_ISR) 261
22.9.5 ProxSense Clock Control Register 1 (PXS_CKCR1) 262
22.9.6 ProxSense Clock Control Register 2 (PXS_CKCR2) 263
22.9.7 Receiver Enable Register for Rx[9:8] (PXS_RXENRH) 264
22.9.8 Receiver Enable Register for Rx[7:0] (PXS_RXENRL) 264
22.9.9 Receiver Control Register 1 for Rx[9:8] (PXS_RXCR1H) 264
22.9.10 Receiver Control Register 1 for Rx[7:0] (PXS_RXCR1L) 264
22.9.11 Receiver Control Register 2 for Rx[9:8] (PXS_RXCR2H) 265
22.9.12 Receiver Control Register 2 for Rx[7:0] (PXS_RXCR2L) 265
22.9.13 Receiver Control Register 3 for Rx[9:8] (PXS_RXCR3H) 265
22.9.14 Receiver Control Register 3 for Rx[7:0] (PXS_RXCR3L) 265
22.9.15 Receiver Inactive State Register for Rx[9:8] (PXS_RXINSRH) 266
22.9.16 Receiver Inactive State Register for Rx[7:0] (PXS_RXINSRL) 266
22.9.17 Transmit Enable Register for Tx[15:8] (PXS_TXENRH) 267
22.9.18 Transmit Enable Register for Tx[7:0] (PXS_TXENRL) 267
22.9.19 Maximum Counter Value Register for Rx[9:8] (PXS_MAXRH) 267
22.9.20 Maximum Counter Value Register for Rx[7:0] (PXS_MAXRL) 267
22.9.21 Maximum Counter Enable Register for Rx[9:8] (PXS_MAXENRH) . . . 268
22.9.22 Maximum Counter Enable Register for Rx[7:0] (PXS_MAXENRL) . . . 268
22.9.23 Receiver Status Register for Rx[9:8] (PXS_RXSRH) 269
22.9.24 Receiver Status Register for Rx[7:0] (PXS_RXSRL) 269
22.9.25 Counter Register for bits [15:8] of Receiver Channel n (PXS_RXnCNTRH) 270
22.9.26 Counter Register for bits [7:0] of Receiver Channel n (PXS_RXnCNTRL) 270
22.9.27 Receiver 0 to 9 Sampling Capacitor (CS) Size Selection Registers 0-9 (PXS_RXnCSSELR) 271
22.9.28 Receiver 0- to 9 Electrode Parasitic Compensation Capacitor (EPCC) Size Selection Registers 0-9 (PXS_RXnEPCCSELR) 271
22.9.29 ProxSense register map and reset values 272

23 Revision history 274

List of tables

Table 1. List of abbreviations . . . 17
Table 2. Interrupt levels . . . 21
Table 3. CPU register map . . . 22
Table 4. CFG_GCR register map . . . 23
Table 5. Block size . . . 33
Table 6. Memory access versus programming method . . . 34
Table 7. Flash register map . . . 39
Table 8. Software priority levels . . . 40
Table 9. Interrupt enabling/disabling inside an ISR . . . 41
Table 10. Vector address map versus software priority bits . . . 46
Table 11. External interrupt sensitivity . . . 47
Table 12. Dedicated interrupt instruction set . . . 47
Table 13. ITC and EXTI register map . . . 55
Table 14. RST register map and reset values . . . 60
Table 15. Peripheral clock gating bits . . . 64
Table 16. CLK register map and reset values . . . 66
Table 17. Low power mode management . . . 69
Table 18. WFE register map . . . 74
Table 19. I/O port configuration summary . . . 77
Table 20. Effect of low power modes on GPIO ports . . . 78
Table 21. GPIO register map . . . 82
Table 22. Register map . . . 83
Table 23. Time base calculation table . . . 86
Table 24. AWU register map . . . 89
Table 25. Beeper register map . . . 92
Table 26. Window watchdog timing example . . . 95
Table 27. Effect of low power modes on WWDG . . . 95
Table 28. WWDG register map and reset values . . . 97
Table 29. Min/Max IWDG timeout (LSI clock frequency = 38 kHz) . . . 99
Table 30. IWDG register map . . . 101
Table 31. Timer characteristics . . . 102
Table 32. Timer feature comparison . . . 102
Table 33. Glossary of internal timer signals . . . 103
Table 34. Counting direction versus encoder signals . . . 141
Table 35. Output control bit for OCx channels with break feature . . . 160
Table 36. TIMx register map . . . 161
Table 37. TIM4 register map . . . 170
Table 38. I^2C interface behavior in low power modes . . . 184
Table 39. I^2C Interrupt requests . . . 184
Table 40. I2C_CCR values for SCL frequency (fMASTER = 16 MHz) . . . 198
Table 41. I^2C register map . . . 199
Table 42. SPI behavior in low power modes . . . 217
Table 43. SPI interrupt requests . . . 218
Table 44. SPI register map and reset values . . . 223
Table 45. Noise detection from sampled data . . . 234
Table 46. Baud rate programming and error calculation . . . 236
Table 47. USART receiver’s tolerance when USART_DIV[3:0] is 0 . . . 236
Table 48. USART receiver’s tolerance when USART_DIV[3:0] is different from 0 . . . 237

Table 49. Frame formats 237
Table 50. USART interface behavior in low power modes 242
Table 51. USART interrupt requests. 242
Table 52. USART register map. 249
Table 53. PXS pins 251
Table 54. Activities during the UP and PASS phases 252
Table 55. Behavior in low power modes. 256
Table 56. ProxSense register map 272
Table 57. Document revision history 274

List of figures

Figure 1. Programming model . . . 19
Figure 2. Stacking order . . . 20
Figure 3. SWIM pin connection . . . 24
Figure 4. STM8TL5xxx Flash program and data EEPROM organization . . . 26
Figure 5. UBC area size definition for touch sensing STM8TL5xxx devices . . . 28
Figure 6. Interrupt processing flowchart . . . 41
Figure 7. Priority decision process . . . 42
Figure 8. Concurrent interrupt management . . . 45
Figure 9. Nested interrupt management . . . 46
Figure 10. Power supply overview . . . 56
Figure 11. Reset circuit . . . 57
Figure 12. Clock structure . . . 61
Figure 13. GPIO block diagram . . . 76
Figure 14. AWU block diagram . . . 84
Figure 15. Beep block diagram . . . 90
Figure 16. Watchdog block diagram . . . 93
Figure 17. Window watchdog timing diagram . . . 95
Figure 18. Independent watchdog block diagram . . . 98
Figure 19. TIMx general block diagram . . . 106
Figure 20. Time base unit . . . 107
Figure 21. 16-bit read sequence for the counter (TIMx_CNTR) . . . 108
Figure 22. Counter in up-counting mode . . . 109
Figure 23. Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2 . . . 110
Figure 24. Counter update event when ARPE=1 (TIMx_ARR preloaded) . . . 110
Figure 25. Counter in down-counting mode . . . 111
Figure 26. Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2 . . . 112
Figure 27. Counter update when ARPE=1 (ARR preloaded), with prescaler = 1 . . . 112
Figure 28. Counter in center-aligned mode . . . 113
Figure 29. Counter timing diagram, CK_PSC divided by 1, TIMx_ARR=06h, ARPE=1 . . . 114
Figure 30. Clock/trigger controller block diagram . . . 115
Figure 31. Control circuit in normal mode, fMASTER divided by 1 . . . 116
Figure 32. TI2 external clock connection example . . . 116
Figure 33. Control circuit in external clock mode 1 . . . 117
Figure 34. External trigger input block . . . 117
Figure 35. Control circuit in external clock mode 2 . . . 118
Figure 36. Control circuit in trigger mode . . . 119
Figure 37. Control circuit in trigger reset mode . . . 119
Figure 38. Control circuit in trigger gated mode . . . 120
Figure 39. Control circuit in external clock mode 2 + trigger mode . . . 121
Figure 40. Timer chaining system implementation example . . . 122
Figure 41. Trigger/master mode selection blocks . . . 122
Figure 42. Master/slave timer example . . . 123
Figure 43. Gating Timer B with OC1REF of Timer A . . . 124
Figure 44. Gating Timer B with the counter enable signal of Timer A (CNT_EN) . . . 125
Figure 45. Triggering Timer B with update event of Timer A (TIMERA-UEV) . . . 126
Figure 46. Triggering Timer B with counter enable CNT_EN of Timer A . . . 126
Figure 47. Triggering Timer A and B with Timer A TI1 input . . . 127
Figure 48. Capture/compare channel 1 main circuit . . . 128
Figure 49. 16-bit read sequence for the TIMx_CCRi register in capture mode . . . 129
Figure 50. Channel input stage block diagram . . . 129
Figure 51. Input stage of TIM 1 channel 1 . . . 130

Figure 52. PWM input signal measurement . . . 131
Figure 53. PWM input signal measurement example . . . 132
Figure 54. Channel output stage block diagram . . . 132
Figure 55. Output stage of channel 1 . . . 133
Figure 56. Output compare mode, toggle on OC1 . . . 134
Figure 57. Edge-aligned counting mode PWM mode 1 waveforms (ARR=8) . . . 135
Figure 58. Center-aligned PWM waveforms (ARR=8) . . . 136
Figure 59. Example of one pulse mode . . . 137
Figure 60. Behavior of outputs in response to a break . . . 139
Figure 61. ETR activation . . . 140
Figure 62. Example of counter operation in encoder interface mode . . . 142
Figure 63. Example of encoder interface mode with IC1 polarity inverted . . . 142
Figure 64. TIM4 block diagram . . . 163
Figure 65. I^2C bus protocol . . . 172
Figure 66. I^2C block diagram . . . 173
Figure 67. Transfer sequence diagram for slave transmitter . . . 175
Figure 68. Transfer sequence diagram for slave receiver . . . 176
Figure 69. Transfer sequence diagram for master transmitter . . . 179
Figure 70. Transfer sequence diagram for master receiver . . . 180
Figure 71. I2C interrupt mapping diagram . . . 185
Figure 72. SPI block diagram . . . 202
Figure 73. Single master/ single slave application . . . 203
Figure 74. Data clock timing diagram . . . 205
Figure 75. TXE/RXNE/BSY behavior in full duplex mode (RXONLY = 0). Case of continuous transfers . . . 210
Figure 76. TXE/RXNE/BSY behavior in slave / full duplex mode (BDM = 0, RXONLY = 0). Case of continuous transfers . . . 210
Figure 77. TXE/BSY in master transmit-only mode (BDM = 0 and RXONLY = 0). Case of continuous transfers . . . 211
Figure 78. TXE/BSY in slave transmit-only mode (BDM = 0 and RXONLY = 0). Case of continuous transfers . . . 212
Figure 79. RXNE behavior in receive-only mode (BDM = 0 and RXONLY = 1). Case of continuous transfers . . . 213
Figure 80. TXE/BSY behavior when transmitting (BDM = 0 and RXLONY = 0). Case of discontinuous transfers . . . 214
Figure 81. USART block diagram . . . 227
Figure 82. Word length programming . . . 228
Figure 83. Configurable STOP bits . . . 229
Figure 84. TC/TXE behavior when transmitting . . . 231
Figure 85. Start bit detection . . . 232
Figure 86. Data sampling for noise detection . . . 234
Figure 87. How to code USART_DIV in the BRR registers . . . 235
Figure 88. Mute mode using Idle line detection . . . 238
Figure 89. Mute mode using address mark detection . . . 239
Figure 90. USART example of synchronous transmission . . . 240
Figure 91. USART data clock timing diagram (M=0) . . . 241
Figure 92. USART data clock timing diagram (M=1) . . . 241
Figure 93. RX data setup/hold time . . . 241
Figure 94. USART interrupt mapping diagram . . . 242
Figure 95. ProxSense block diagram . . . 252

1 Memory and register map

For details on the memory map, I/O port hardware register map and CPU/SWIM/debug module/interrupt controller registers, refer to the product datasheets.

1.1 Register description abbreviations

In the register descriptions of each chapter in this reference manual, the following abbreviations are used:

Table 1. List of abbreviations

Abbreviation	Description
read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
write only (w)	Software can only write to this bit. Reading the bit returns a meaningless value.
read/write once (rwo)	Software can only write once to this bit but can read it at any time. Only a reset can return this bit to its reset value.
read/clear (rc_w1)	Software can read and clear this bit by writing 1. Writing ‘0’ has no effect on the bit value.
read/clear (rc_w0)	Software can read and clear this bit by writing 0. Writing ‘1’ has no effect on the bit value.
read/set (rs)	Software can read and set this bit. Writing ‘0’ has no effect on the bit value.
read/clear by read (rc_r)	Software can read this bit. Reading this bit automatically clears it to ‘0’. Writing ‘0’ has no effect on the bit value.
Reserved (Res.)	Reserved bit, must be kept at reset value.

2 Central processing unit (CPU)

2.1 Introduction

The CPU has an 8-bit architecture. Six internal registers allow efficient data manipulations. The CPU is able to execute 80 basic instructions. It features 20 addressing modes and can address six internal registers. For the complete description of the instruction set, refer to the STM8 microcontroller family programming manual (PM0044).

2.2 CPU registers

The six CPU registers are shown in the programming model in *Figure 1*. Following an interrupt, the registers are pushed onto the stack in the order shown in *Figure 2*. They are popped from stack in the reverse order. The interrupt routine must therefore handle it, if needed, through the POP and PUSH instructions.

2.2.1 Description of CPU registers

Accumulator (A)

The accumulator is an 8-bit general purpose register used to hold operands and the results of the arithmetic and logic calculations as well as data manipulations.

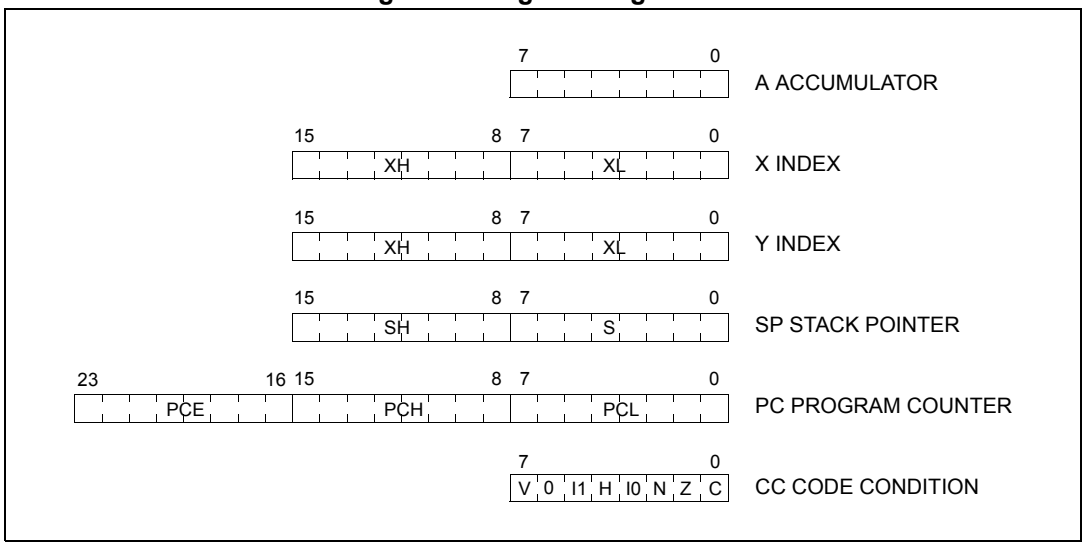
Index registers (X and Y)

These are 16-bit registers used to create effective addresses. They may also be used as a temporary storage area for data manipulations and have an inherent use for some instructions (multiplication/division). In most cases, the cross assembler generates a PRECODE instruction (PRE) to indicate that the following instruction refers to the Y register.

Program counter (PC)

The program counter is a 24-bit register used to store the address of the next instruction to be executed by the CPU. It is automatically refreshed after each processed instruction. As a result, the STM8 core can access up to 16 Mbytes of memory.

Figure 1. Programming model



Stack pointer (SP)

The stack pointer is a 16-bit register. It contains the address of the next free location of the stack. Depending on the product, the most significant bits can be forced to a preset value.

The stack is used to save the CPU context on subroutine calls or interrupts. The user can also directly use it through the POP and PUSH instructions.

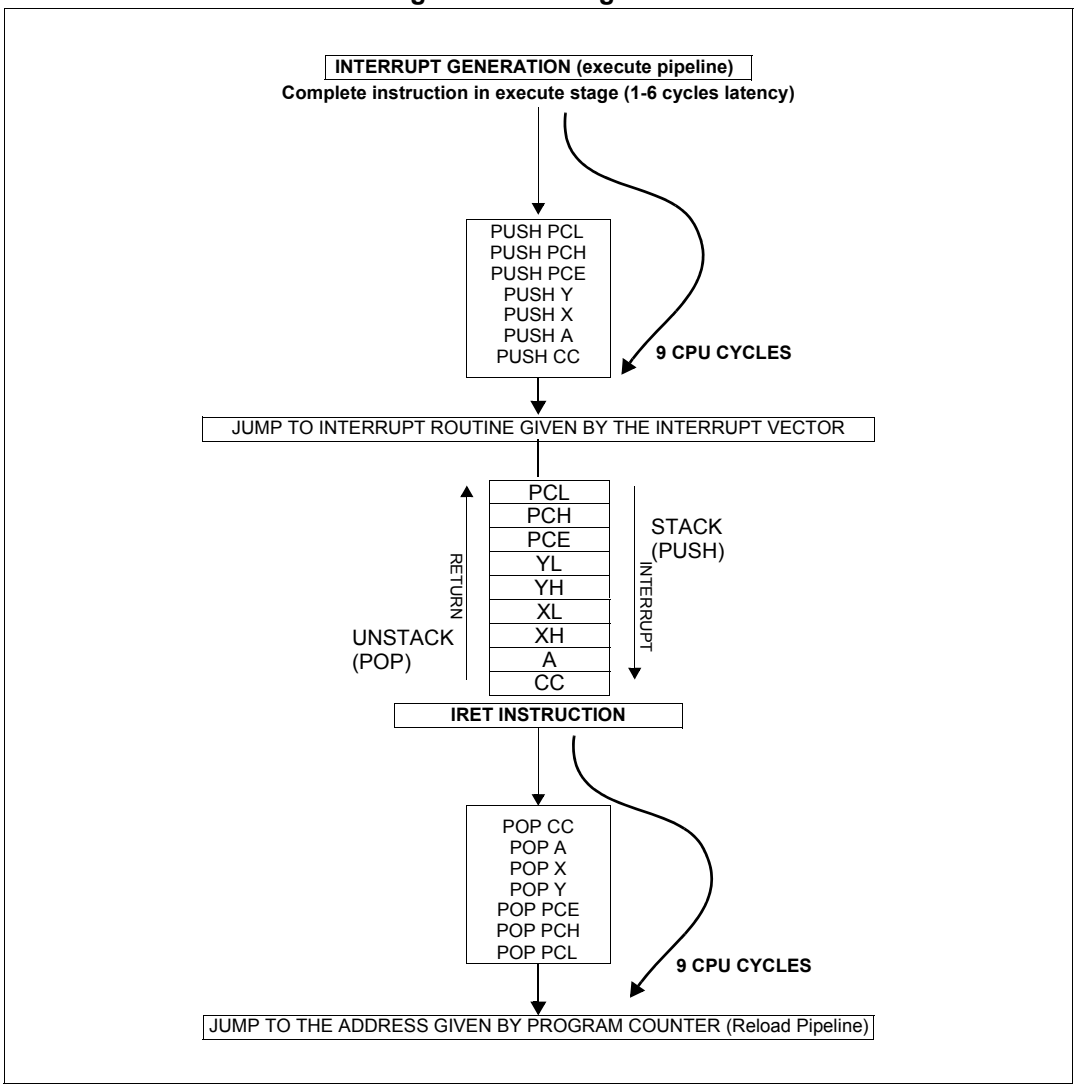
The stack pointer can be initialized by the startup function provided with the C compiler. For applications written in C language, the initialization is then performed according to the address specified in the linker file for C users. If you use your own linker file or startup file, make sure the stack pointer is initialized properly (with the address given in the datasheets). For applications written in assembler, you can use either the startup function provided by ST or write your own by initializing the stack pointer with the correct address.

The stack pointer is decremented after data has been pushed onto the stack and incremented after data is popped from the stack. It is up to the application to ensure that the lower limit is not exceeded.

A subroutine call occupies two or three locations. An interrupt occupies nine locations to store all the internal registers (except SP). For more details refer to *Figure 2*.

Note: The WFI/HALT instructions save the context in advance. If an interrupt occurs while the CPU is in one of these modes, the latency is reduced.

Figure 2. Stacking order



Condition code register (CC)

The condition code register is an 8-bit register which indicates the result of the instruction just executed as well as the state of the processor. The 6th bit (MSB) of this register is reserved. These bits can be individually tested by a program and specified action taken as a result of their state. The following paragraphs describe each bit:

- V: Overflow

When set, V indicates that an overflow occurred during the last signed arithmetic operation, on the MSB result bit. See the INC, INCW, DEC, DECW, NEG, NEGW, ADD, ADDW, ADC, SUB, SUBW, SBC, CP, and CPW instructions.

- I1: Interrupt mask level 1

The I1 flag works in conjunction with the I0 flag to define the current interruptability level as shown in *Table 2*. These flags can be set and cleared by software through the RIM, SIM, HALT, WFI, IRET, TRAP, and POP instructions and are automatically set by hardware when entering an interrupt service routine.

Table 2. Interrupt levels

Interruptability	Priority	I1	I0
Interruptable main	Lowest	1	0
Interruptable level 1		0	1
Interruptable level 2		0	0
Non interruptable	Highest	1	1

- H: Half carry bit

The H bit is set to 1 when a carry occurs between the bits 3 and 4 of the ALU during an ADD or ADC instruction. The H bit is useful in BCD arithmetic subroutines.

- I0: Interrupt mask level 0

See Flag I1.

- N: Negative

When set to 1, this bit indicates that the result of the last arithmetic, logical or data manipulation is negative (i.e. the most significant bit is a logic 1).

- Z: Zero

When set to 1, this bit indicates that the result of the last arithmetic, logical or data manipulation is zero.

- C: Carry

When set, C indicates that a carry or borrow out of the ALU occurred during the last arithmetic operation on the MSB operation result bit. This bit is also affected during bit test, branch, shift, rotate and load instructions. See the ADD, ADC, SUB, and SBC instructions.

In a division operation, C indicates if trouble occurred during execution (quotient overflow or zero division). See the DIV instruction.

In bit test operations, C is the copy of the tested bit. See the BTJF and BTJT instructions. In shift and rotate operations, the carry is updated. See the RRC, RLC, SRL, SLL, and SRA instructions.

This bit can be set, reset or complemented by software using the SCF, RCF, and CCF instructions.

Example: Addition

$B5 + $94 = "C" + $49 = $149

	C	7						0
	0	1	0	1	1	0	1	0	1
	C	7						0
+	0	1	0	0	1	0	1	0	0
	C	7						0
=	1	0	1	0	0	1	0	0	1

2.2.2 STM8 CPU register map

The CPU registers are mapped in the STM8 address space as shown in *Table 3*. These registers can only be accessed by the debug module but not by memory access instructions executed in the core.

Table 3. CPU register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	A	MSB	-	-	-	-	-	-	LSB
0x01	PCE	MSB	-	-	-	-	-	-	LSB
0x02	PCH	MSB	-	-	-	-	-	-	LSB
0x03	PCL	MSB	-	-	-	-	-	-	LSB
0x04	XH	MSB	-	-	-	-	-	-	LSB
0x05	XL	MSB	-	-	-	-	-	-	LSB
0x06	YH	MSB	-	-	-	-	-	-	LSB
0x07	YL	MSB	-	-	-	-	-	-	LSB
0x08	SPH	MSB	-	-	-	-	-	-	LSB
0x09	SPL	MSB	-	-	-	-	-	-	LSB
0x0A	CC	V	0	I1	H	I0	N	Z	C

2.3 Global configuration register (CFG_GCR)

2.3.1 Activation level

The MCU activation level is configured by programming the AL bit in the CFG_GCR register.

For information on the use of this bit refer to *Section 5.4: Activation level/low power mode control on page 44*.

2.3.2 SWIM disable

By default, after an MCU reset, the SWIM pin is configured to allow communication with an external tool for debugging or Flash/EEPROM programming. This pin can be configured by the application for use as a general purpose I/O. This is done by setting the SWD bit in the CFG_GCR register.

2.3.3 Description of global configuration register (CFG_GCR)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
					Reserved	AL	SWD
						rw	rw

Bits 7:2 Reserved

Bit 1 **AL**: Activation level

This bit is set and cleared by software. It configures main or interrupt-only activation.

0: Main activation level. An IRET instruction causes the context to be retrieved from the stack and the main program continues after the WFI instruction.

1: Interrupt-only activation level. An IRET instruction causes the CPU to go back to WFI/Halt mode without restoring the context.

Bit 0 **SWD**: SWIM disable

0: SWIM mode enabled

1: SWIM mode disabled

When SWIM mode is enabled, the SWIM pin cannot be used as general purpose I/O.

2.3.4 Global configuration register map and reset values

The CFG_GCR is mapped in the STM8 address space. Refer to the corresponding datasheets for the base address.

Table 4. CFG_GCR register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	CFG_GCR Reset value	- 0	- 0	- 0	- 0	- 0	- 0	AL 0	SWD 0

3 Single wire interface module (SWIM) and debug module (DM)

3.1 Introduction

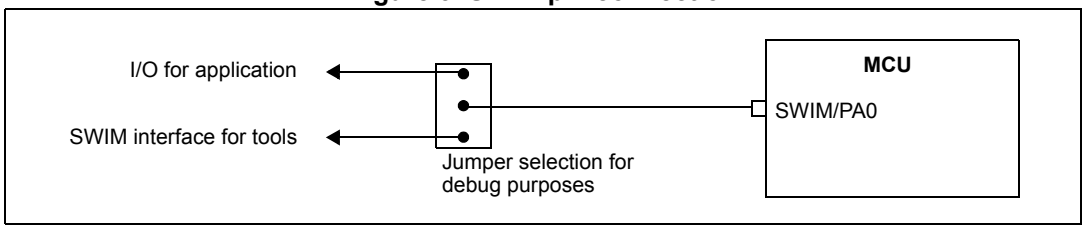
In-circuit debugging mode or in-circuit programming mode are managed through a single wire hardware interface featuring ultrafast memory programming. Coupled with an in-circuit debugging module, it also offers a non-intrusive emulation mode, making the in-circuit debugger extremely powerful, close in performance to a full-featured emulator.

3.2 Main features

- Based on an asynchronous, high sink (8 mA), open-drain, bidirectional communication.
- Allows reading or writing any part of memory space.
- Access to CPU registers (A, X, Y, CC, SP). They are memory mapped for read or write access.
- Non intrusive read/write on the fly to the RAM and peripheral registers.
- Device reset capability with status flag in the *Reset status register (RST_SR)*.

SWIM pin can be used as a standard I/O with some restrictions if you also want to use it for debug. The most secure way is to provide on the PCB a strap option.

Figure 3. SWIM pin connection



3.3 SWIM modes

After a power-on reset, the SWIM is reset and enters OFF mode.

1. **OFF**: Default state after power-on reset. The SWIM pin cannot be used by the application as an I/O.
2. **I/O**: This state is entered by software writing to the SWD bit in the *Global configuration register (CFG_GCR)*. In this state, the SWIM pin can be used by the application as a standard I/O pin. In case of a reset, the SWIM goes back to OFF mode.
3. **SWIM**: This state is entered when a specific sequence is performed on the SWIM pin. In this state, the SWIM pin is used by the host tool to control the STM8 with 3 commands (SRST system reset, ROTF read on the fly, WOTF write on the fly).

Note: Refer to the STM8 SWIM communication Protocol and Debug Module User Manual for a description of the SWIM and Debug module (DM) registers. Refer also to the Description of global configuration register (CFG_GCR) on page 23.

4 Flash program memory and data EEPROM

4.1 Introduction

The embedded Flash program memory and data EEPROM memories are controlled by a common set of registers. Using these registers, the application can program or erase memory contents and set write protection. The application can also program the device option bytes.

4.2 Glossary

- Block

  A block is a set of bytes that can be programmed or erased in one single programming operation. Operations that are performed at block level are faster than standard programming and erasing. Refer to *Table 5* for the details on block size.
- Page

  A page is a set of blocks.

  Dedicated option bytes can be used to configure, by increments of one page, the size of the user boot code, data EEPROM and proprietary code.

4.3 Main Flash memory features

- The EEPROM for **STM8TL5xxx touch sensing devices** is divided into two memory arrays (see *Section 4.4: Memory organization* for details on memory mapping):
  - Up to 16 Kbytes of embedded Flash program including up to 2 Kbytes of data EEPROM. Data EEPROM and Flash program areas can be write protected independently by using the memory access security mechanism (MASS).
  - 64 option bytes (one block) of which 5 bytes are already used for the device.
- Programming modes
  - Byte programming and automatic fast byte programming (without erase operation)
  - Word programming
  - Block programming and fast block programming mode (without erase operation)
  - Interrupt generation on end of program/erase operation and on illegal program operation.
- In-application programming (IAP) and in-circuit programming (ICP) capabilities
- Protection features
  - Memory readout protection (ROP)
  - Program memory write protection with memory access security system (MASS keys)
  - Data memory write protection with memory access security system (MASS keys)
  - Programmable write protected user boot code area (UBC).

4.4 Memory organization

The STM8TL5xxx touch sensing Flash program memory is divided into 256 pages of 64 bytes each. It is organized in 32-bit words (4 bytes per word).

The memory array is divided into four areas:

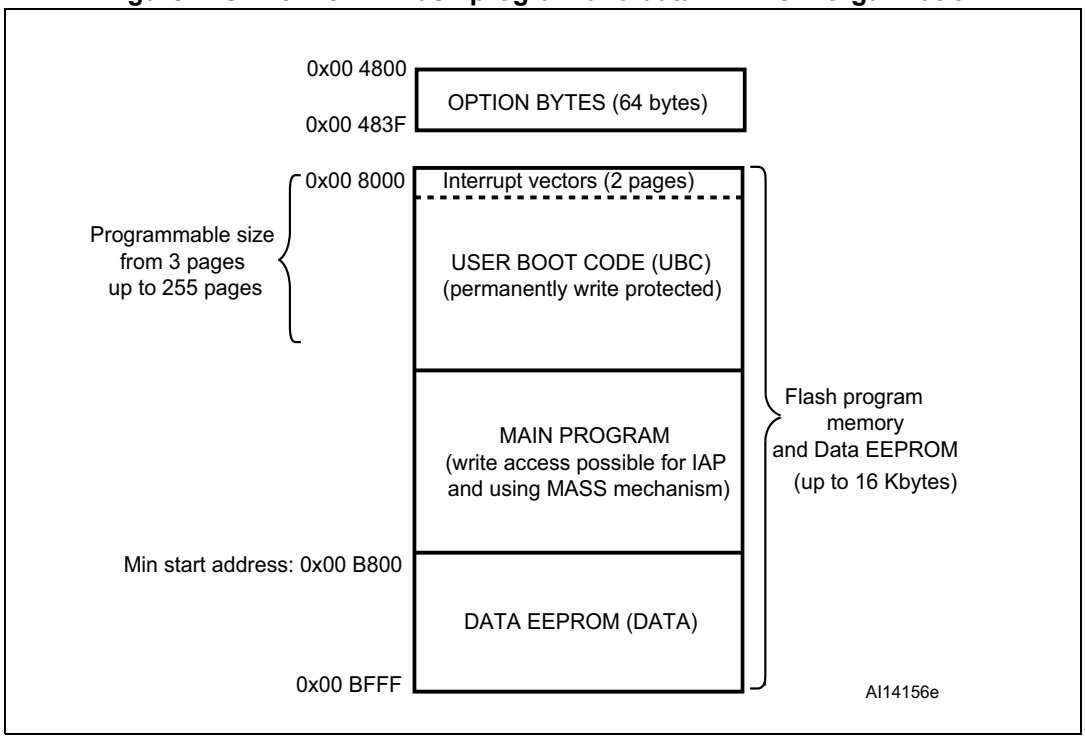
- The proprietary code area (PCODE)
- The user boot code area (UBC)
- The main program area
- The data EEPROM (DATA)

The first two pages of the Flash program memory (starting from address 0x00 8000) contain the interrupt vectors.

The devices also feature one block of option bytes (64 bytes) located a separate memory array.

See *Figure 4* for a description of the memory organization.

Figure 4. STM8TL5xxx Flash program and data EEPROM organization



4.4.1 Proprietary code area (PCODE)

The proprietary code area (PCODE) can be used to protect proprietary software libraries used to drive peripherals.

The size of the PCODE area can be configured through the PCODE option byte (PCODESIZE) in ICP mode (using the SWIM interface). This option byte specifies the number of pages (64-byte granularity) allocated for the PCODE area starting from address 0x00 8000. Once programmed, the PCODE option byte cannot be erased, and the size of the PCODE area remains fixed.

The minimum meaningful size of the PCODE area is 1 page ranging from address 0x8080 to 0x80BF while the PCODESIZE is set to the value 3. The maximum size of the PCODE area is 253 pages ranging from address 0x00 8080 to 0x00 BFBF (while the PCODESIZE is 0xFF). In this case, only one page is left either for the main program area or for data EEPROM. While the PCODE is enabled the TRAP interrupt vector is write protected.

The PCODE area is automatically readout protected except for the TRAP interrupt vector (see *Section 4.5.1: Readout protection*). The readout protection cannot be disabled in this area. This means that the content of the PCODE area cannot be read or modified.

The PCODE area can be accessed only through the TRAP vector.

4.4.2 User boot area (UBC)

The user boot area (UBC) contains the reset and the interrupt vectors. It can be used to store the IAP and communication routines. The UBC area has a second level of protection to prevent unintentional erasing or modification during IAP programming. This means that it is always write protected and the write protection cannot be unlocked using the MASS keys.

The size of the UBC area can be obtained by reading the UBC option byte.

The size of the UBC area can be configured in ICP mode (using the SWIM interface) through the UBC option byte. The UBC option byte specifies the number of pages allocated for the UBC area starting from address 0x00 8000.

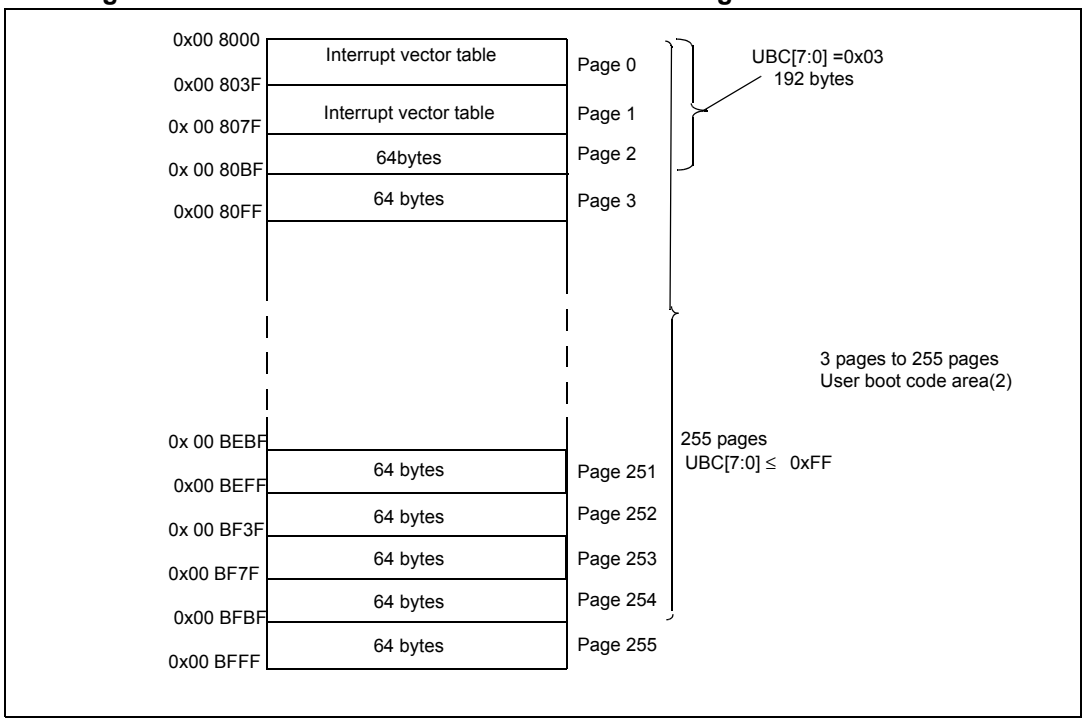
The minimum meaningful size of the UBC area is of 3 pages of which 2 are used to store the interrupt vectors.

Note: When a PCODE area has been defined, the minimum size of the UBC should be PCODESIZE+1. The portion of the UBC available to store the boot code is the area located between the end of the PCODE area and the end of the defined UBC area.

The maximum size of the boot area is 255 pages ranging from address 0x00 8000 to 0x00 BFBF (including the interrupt vectors). In this case, only one page is left either for the main program area or for data EEPROM.

Refer to *Figure 5* for a description of the UBC area memory mapping and to the option byte section in the datasheets for more details on the UBC option byte.

Figure 5. UBC area size definition for touch sensing STM8TL5xxx devices



1. UBC[7:0]= 0x00 means no memory space is allocated for the UBC area.
2. Page 255 is reserved for the main program area or data EEPROM.

4.4.3 Data EEPROM (DATA)

The data EEPROM area can be used to store application data. By default, the DATA area is write protected to prevent unintentional modification when the main program is updated in IAP mode. The write protection can be unlocked only by using a specific MASS key sequence (refer to *Enabling write access to the DATA area*).

The size of the DATA area can be configured through the DATA option byte (DATASIZE) in ICP mode. This option byte specifies the number of pages (64 bytes) starting down from address 0x00 BFFF.

The maximum size of the DATA area is of 2 Kbytes, corresponding to start address 0x00 B800 (see *Figure 4*) i.e. up to 32 pages of 64 bytes can be defined as data EEPROM.

4.4.4 Main program area

The main program is the area which starts at the end of the UBC or PCODE and ends at address 0x00 BFFF. It is used to store the application code (see *Figure 4*).

4.4.5 Option bytes

The option bytes are used to configure device hardware features and memory protection. They are located in a dedicated memory array of one block.

The option bytes can be modified only in ICP/SWIM mode with OPT bit of the FLASH_CR2 register set to 1 (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*).

Refer to the option byte section in the datasheet for more information on option bytes, and to the STM8 SWIM protocol and debug module user manual (UM0470) for details on how to program them.

4.5 Memory protection

4.5.1 Readout protection

Readout protection is selected by programming the ROP option byte to 0xAA. When readout protection is enabled, reading or modifying the Flash program memory and DATA area in ICP mode (using the SWIM interface) is forbidden, whatever the write protection settings.

Even if no protection can be considered as totally unbreakable, the readout feature provides a very high level of protection for a general purpose microcontroller.

The readout protection can be disabled on the program memory, UBC, PCODE and DATA areas, by reprogramming the ROP option byte in ICP mode. In this case, the Flash program memory, the DATA area and the option bytes are automatically erased and the device can be reprogrammed.

By default, the PCODE area is always readout protected except for the interrupt vector TRAP (see *Section 4.4.1: Proprietary code area (PCODE)*).

Refer to *Table 6: Memory access versus programming method* for details on memory access when readout protection is enabled or disabled.

4.5.2 Memory access security system (MASS)

After reset, the main program and DATA areas (when they exist) are protected against unintentional write operations. They must be unlocked before attempting to modify their content. This unlock mechanism is managed by the memory access security system (MASS).

The UBC area specified in the UBC option byte is always write protected (see *Section 4.4.2: User boot area (UBC)*).

Once the memory has been modified, it is recommended to enable the write protection again to protect the memory content against corruption.

Enabling write access to the main program memory

After a device reset, it is possible to disable the main program memory write protection by writing consecutively two values called MASS keys to the FLASH_PUKR register (see *Section 4.8.3: Flash program memory unprotecting key register (FLASH_PUKR)*). These programmed keys are then compared to two hardware key values:

- First hardware key: 0b0101 0110 (0x56)
- Second hardware key: 0b1010 1110 (0xAE)

The following steps are required to disable write protection of the main program area:

1. Write a first 8-bit key into the FLASH_PUKR register. When this register is written for the first time after a reset, the data bus content is not latched into the register, but compared to the first hardware key value (0x56).
2. If the key available on the data bus is incorrect, the FLASH_PUKR register remains locked until the next reset. Any new write commands sent to this address are discarded.
3. If the first hardware key is correct when the FLASH_PUKR register is written for the second time, the data bus content is still not latched into the register, but compared to the second hardware key value (0xAE).
4. If the key available on the data bus is incorrect, the write protection on program memory remains locked until the next reset. Any new write commands sent to this address is discarded.
5. If the second hardware key is correct, the main program memory is write unprotected and the PUL bit of the FLASH_IAPSR is set (see *Section 4.8.5: Flash status register (FLASH_IAPSR)*) register.

Before starting programming, the application must verify that PUL bit is effectively set. The application can choose, at any time, to disable again write access to the Flash program memory by clearing the PUL bit.

Enabling write access to the DATA area

After a device reset, it is possible to disable the DATA area write protection by writing consecutively two values called MASS keys to the FLASH_DUKR register (see *Section 4.8.6: Flash register map and reset values*). These programmed keys are then compared to two hardware key values:

- First hardware key: 0b1010 1110 (0xAE)
- Second hardware key: 0b0101 0110 (0x56)

The following steps are required to disable write protection of the DATA area:

1. Write a first 8-bit key into the FLASH_DUKR register. When this register is written for the first time after a reset, the data bus content is not latched into the register, but compared to the first hardware key value (0xAE).
2. If the key available on the data bus is incorrect, the application can re-enter two MASS keys to try unprotecting the DATA area.
3. If the first hardware key is correct, the FLASH_DUKR register is programmed with the second key. The data bus content is still not latched into the register, but compared to the second hardware key value (0x56).
4. If the key available on the data bus is incorrect, the data EEPROM area remains write protected until the next reset. Any new write command sent to this address is ignored.
5. If the second hardware key is correct, the DATA area is write unprotected and the DUL bit of the FLASH_IAPSR register is set (see *Section 4.8.5: Flash status register (FLASH_IAPSR)*).

Before starting programming, the application must verify that the DATA area is not write protected by checking that the DUL bit is effectively set. The application can choose, at any time, to disable again write access to the DATA area by clearing the DUL bit.

4.6 Memory programming

The main program memory, and the DATA area must be unlocked before attempting to perform any program operation. The unlock mechanism depends on the memory area to be programmed as described in *Section 4.5.2: Memory access security system (MASS)*.

Note: The PCODE area is always readout protected. The only way to reprogram it is to reset the ROP option byte, thus erasing the Flash program memory, DATA area, and option bytes.

4.6.1 Byte programming

The main program memory and the DATA area can be programmed at byte level. To program one byte, the application writes directly to the target address.The application stops for the duration of the byte program operation.

To erase a byte, simply write 0x00 at the corresponding address.

The application can read the FLASH_IAPSR register to verify that the programming or erasing operation has been correctly executed:

- EOP flag is set after a successful programming operation
- WR_PG_DIS is set when the software has tried to write to a protected page. In this case, the write procedure is not performed.

As soon as one of these flags are set, a Flash interrupt is generated if it has been previously enabled by setting the IE bit of the FLASH_CR1 register.

Automatic fast byte programming

The programming duration can vary according to the initial content of the target address. If the word (4 bytes) containing the byte to be programmed is not empty, the whole word is automatically erased before the program operation. On the contrary if the word is empty, no erase operation is performed and the programming time is shorter (see t_{PROG} in *Table "Flash program memory" in the datasheet*).

However, the programming time can be fixed by setting the FIX bit of the FLASH_CR1 register to force the program operation to systematically erase the byte whatever its content (see *Section 4.8.1: Flash control register 1 (FLASH_CR1)*). The programming time is consequently fixed and equal to the sum of the erase and write time (see t_{PROG} in *Table "Flash program memory" in the datasheet*).

Note: To write a byte fast (no erase), the whole word (4 bytes) into which it is written must be erased beforehand. Consequently, It is not possible to do two fast writes to the same word (without an erase before the second write): The first write will be fast but the second write to the other byte will require an erase.

4.6.2 Word programming

A word write operation allows an entire 4-byte word to be programmed in one shot, thus minimizing the programming time.

As for byte programming, word operation is available both for the main program memory and data EEPROM.

To program a word, the WPRG bit in the FLASH_CR2 register must be previously set to enable word programming mode (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*). Then, the 4 bytes of the word to be programmed must be loaded starting

with the first address. The programming cycle starts automatically when the 4 bytes have been written.

As for byte operation, the EOP and the WR_PG_DIS control flags of FLASH_IAPSR, together with the Flash interrupt, can be used to determine if the operation has been correctly completed.

4.6.3 Block programming

Block program operations are much faster than byte or word program operations. In a block program operation, a whole block is programmed or erased in a single programming cycle. Refer to *Table 5* for details on the block size according to the devices.

Block operations can be performed both to the main program memory and DATA area. They are executed totally from RAM.

There are three possible block operations:

- Block programming, also called standard block programming: The block is automatically erased before being programmed.
- Fast block programming: No previous erase operation is performed.
- Block erase

During block programming, interrupts are masked by hardware.

Standard block programming

A standard block program operation allows a whole block to be written in one shot. The block is automatically erase before being programmed.

To program a whole block in standard mode, the PRG bit in the FLASH_CR2 register must be previously set to enable standard block programming (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*). Then, the block of data to be programmed must be loaded sequentially to the destination addresses in the main program memory or DATA area. This causes all the bytes of data to be latched. To start programming the whole block, all bytes of data must be written. All bytes written in a programming sequence must be in the same block. This means that they must have the same high address: Only the six least significant bits of the address can change. When the last byte of the target block is loaded, the programming starts automatically. It is preceded by an automatic erase operation of the whole block.

The EOP and the WR_PG_DIS control flags of the FLASH_IAPSR together with the Flash interrupt can be used to determine if the operation has been correctly completed.

Fast block programming

Fast block programming allows programming without first erasing the memory contents. Fast block programming is therefore twice as fast as standard programming.

This mode is intended only for programming parts that have already been erased. It is very useful for programming blank parts with the complete application code, as the time saving is significant.

Fast block programming is performed by using the same sequence as standard block programming. To enable fast block programming mode, the FPRG bit of the FLASH_CR2 registers must be previously set.

The EOP and WR_PG_DIS bits of the FLASH_IAPSR register can be checked to determine if the fast block programming operation has been correctly completed.

Caution: The data programmed in the block are not guaranteed when the block is not blank before the fast block program operation.

Block erasing

A block erase allows a whole block to be erased.

To erase a whole block, the ERASE bit in the FLASH_CR2 register must be previously set to enable block erasing (see *Section 4.8.2: Flash control register 2 (FLASH_CR2)*). The block is then erased by writing ‘0x00 00 00 00’ to any word inside the block. The word start address must end with ‘0’, ‘4’, ‘8’, or ‘C’.

The EOP and the WR_PG_DIS control flags of the FLASH_IAPSR together with the Flash interrupt can be used to determine if the operation has been correctly completed.

Table 5. Block size

STM8 microcontroller family	Block size
STM8TL5xxx touch sensing	64 bytes

4.7 ICP and IAP

The in-circuit programming (ICP) method is used to update the entire content of the memory, using the SWIM interface to load the user application into the microcontroller. ICP offers quick and efficient design iterations and eliminates unnecessary package handling or socketing of devices. The SWIM interface (single wire interface module) uses the SWIM pin to connect to the programming tool.

In contrast to the ICP method, in-application programming (IAP) can use any communication interface supported by the microcontroller (I/Os, I^2C, SPI, USART...) to download the data to be programmed in the memory. IAP allows the Flash program memory content to be reprogrammed during application execution. Nevertheless, part of the application must have been previously programmed in the Flash program memory using ICP.

Refer to the STM8TL5xxx Flash programming manual (PM0212) and STM8 SWIM protocol and debug manual (UM0470) for more information on programming procedures.

Table 6. Memory access versus programming method(1)

Mode	ROP	Memory area	Access from core
User mode, IAP, and bootloader	Readout protection enabled	Interrupt vectors except for TRAP	R/W(2)/E
		TRAP	R/W(3)/E
		Proprietary code area (PCODE)	R/E(6)
		User boot code area (UBC)	R/E
		Main program	R/W/E(4)
		Data EEPROM area (DATA)	R/W(5)
		Option bytes	R
	Readout protection disabled	Interrupt vectors except for TRAP	R/W(2)/E
		TRAP	R/W(3)/E
		Proprietary code area (PCODE)	R/E(6)
		User boot code area (UBC)	R/E(7)
		Main program	R/W/E(4)
		Data EEPROM area (DATA)	R/W(5)
		Option bytes	R/W(8)
SWIM active (ICP mode)	Readout protection enabled	Interrupt vectors except for TRAP	P
		TRAP	P
		Proprietary code area (PCODE)	P(6)
		User boot code area (UBC)	P
		Main program	P
		Data EEPROM area (DATA)	P
		Option bytes	P/W_{ROP}(9)
	Readout protection disabled	Interrupt vectors except for TRAP	R/W(2)/E
		TRAP	R/W/E
		Proprietary code area (PCODE)	R/E(6)
		User boot code area (UBC)	R/E(7)
		Main program	R/W/E(4)
		Data EEPROM area (DATA)	R/W(5)
		Option bytes	R/W(7)

1. R/W/E = Read, write, and execute
   R/E = Read and execute (write operation forbidden)
   R = Read (write and execute operations forbidden)
   P = The area cannot be accessed (read, execute and write operations forbidden)
   P/W_{ROP} = Protected, write forbidden except for ROP option byte.
2. When no UBC area has been defined, the interrupt vectors, except for TRAP, can be modified in user/IAP mode.
3. If a PCODE area has been defined, the TRAP vector cannot be modified in user and IAP mode, otherwise TRAP follows the same rules as other interrupt vectors.
4. The Flash program memory is write protected (locked) until the correct MASS key is written in the FLASH_PUKR. It is possible to lock the memory again by resetting the PUL bit in the FLASH_IAPSR register. Unlocking can only be done once between two resets. If incorrect keys are provided, the device must be reset and new keys programmed.

5. The data memory is write protected (locked) until the correct MASS key is written in the FLASH_DUKR. It is possible to lock the memory again by resetting the DUL bit in the FLASH_IAPSR register. If incorrect keys are provided, another key program sequence can be performed without resetting the device.
6. The PCODE area can be read and executed only in privileged mode through TRAP vectors. The PCODE cannot be directly accessed through the SWIM.
7. To program the UBC area the application must first clear the UBC option byte.
8. The option bytes are write protected (locked) until the correct MASS key is written in the FLASH_DUKR (with OPT set to ‘1’). It is possible to lock the memory again by resetting the DUL bit in the FLASH_IAPSR register. If incorrect keys are provided, another key program sequence can be performed without resetting the device.
9. When ROP is removed, the whole memory is erased, including option bytes.

4.8 Flash registers

4.8.1 Flash control register 1 (FLASH_CR1)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						IE	FIX
						rw	rw

Bits 7:2 Reserved

Bit 1 **IE**: Flash Interrupt enable
This bit is set and cleared by software.
0: Interrupt disabled
1: Interrupt enabled. An interrupt is generated if the EOP or WR_PG_DIS flag in the FLASH_IAPSR register is set.

Bit 0 **FIX**: Fixed Byte programming time
This bit is set and cleared by software.
0: Standard programming time of (1/2 t_{prog}) if the memory is already erased and t_{prog} otherwise.
1: Programming time fixed at t_{prog}.

4.8.2 Flash control register 2 (FLASH_CR2)

Address offset: 0x01
Reset value: 0x00

7	6	5	4	3	2	1	0
OPT	WPRG	ERASE	FPRG	Reserved			PRG
rw	rw	rw	rw				rw

Bit 7 **OPT**: Write option bytes
This bit is set and cleared by software.
0: Write access to option bytes disabled
1: Write access to option bytes enabled

Bit 6 **WPRG**: Word programming
This bit is set by software and cleared by hardware when the operation is completed.
0: Word program operation disabled
1: Word program operation enabled

Bit 5 **ERASE**(1): Block erasing
This bit is set by software and cleared by hardware when the operation is completed.
0: Block erase operation disabled
1: Block erase operation enabled

Bit 4 **FPRG**(1)**:** Fast block programming
This bit is set by software and cleared by hardware when the operation is completed.
0: Fast block program operation disabled
1: Fast block program operation enabled

Bits 3:1 Reserved

Bit 0 **PRG**: Standard block programming
This bit is set by software and cleared by hardware when the operation is completed.
0: Standard block programming operation disabled
1: Standard block programming operation enabled (automatically first erasing)

1. The ERASE and FPRG bits are locked when the memory is busy.

4.8.3 Flash program memory unprotecting key register (FLASH_PUKR)

Address offset: 0x02
Reset value: 0x00

7	6	5	4	3	2	1	0
MASS_PRG KEYS							
rw							

Bits 7:0 **PUK [7:0]**: Main program memory unlock keys
This byte is written by software (all modes). It returns 0x00 when read.
Refer to *Enabling write access to the main program memory on page 29* for the description of main program area write unprotection mechanism.

4.8.4 Data EEPROM unprotection key register (FLASH_DUKR)

Address offset: 0x03
Reset value: 0x00

7	6	5	4	3	2	1	0
MASS_DATA KEYS							
rw							

Bits 7:0 **DUK[7:0]**: Data EEPROM write unlock keys

This byte is written by software (all modes). It returns 0x00 when read.

Refer to *Enabling write access to the DATA area on page 30* for the description of main program area write unprotection mechanism.

4.8.5 Flash status register (FLASH_IAPSR)

Address offset: 0x04
Reset value: 0xX0 where X is undefined

7	6	5	4	3	2	1	0
Reserved				DUL	EOP	PUL	WR_PG_DIS
				rc_w0	rc_r	rc_w0	rc_r

Bits 7:4 Reserved

Bit 3 **DUL**: Data EEPROM area unlocked flag

This bit is set by hardware and cleared by software by programming it to 0.

0: Data EEPROM area write protection enabled

1: Data EEPROM area write protection has been disabled by writing the correct MASS keys

Bit 2 **EOP**: End of programming (write or erase operation) flag

This bit is set by hardware. It is cleared by software by reading the register, or when a new write/erase operation starts.

0: No EOP event occurred

1: An EOP operation occurred. An interrupt is generated if the IE bit is set in the FLASH_CR1 register.

Bit 1 **PUL**: Flash Program memory unlocked flag

This bit is set by hardware and cleared by software by programming it to 0.

0: Write protection of main Program area enabled

1: Write protection of main Program area has been disabled by writing the correct MASS keys.

Bit 0 **WR_PG_DIS**: Write attempted to protected page flag

This bit is set by hardware and cleared by software by reading the register.

0: No WR_PG_DIS event occurred

1: A write attempt to a write protected page occurred. An interrupt is generated if the IE bit is set in the FLASH_CR1 register.

4.8.6 Flash register map and reset values

For details on the Flash register boundary addresses, refer to the general hardware register map in the datasheets.

Table 7. Flash register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	FLASH_CR1 Reset value	- 0	- 0	- 0	- 0	- 0	- 0	IE 0	FIX 0
0x01	FLASH_CR2 Reset value	OPT 0	WPRG 0	ERASE 0	FPRG 0	- 0	- 0	- 0	PRG 0
0x02	FLASH_PUKR Reset value	PUK7 0	PUK6 0	PUK5 0	PUK4 0	PUK3 0	PUK2 0	PUK1 0	PUK0 0
0x03	FLASH_DUKR Reset value	DUK7 0	DUK6 0	DUK5 0	DUK4 0	DUK3 0	DUK2 0	DUK1 0	DUK0 0
0x04	FLASH_IAPSR Reset value	- x	- x	- x	- x	DUL 0	EOP 0	PUL 0	WR_PG_DIS 0

5 Interrupt controller (ITC)

5.1 ITC introduction

- Management of hardware interrupts
  - Peripheral interrupt capability
- Management of software interrupt (TRAP)
- Nested or concurrent interrupt management with flexible interrupt priority and level management:
  - Up to 4 software programmable nesting levels
  - 22 interrupt vectors fixed by hardware
  - 2 non maskable events: RESET, TRAP

This interrupt management is based on:

- Bit I1 and I0 of the CPU Condition Code register (CCR)
- Software priority registers (ITC_SPRx)
- Reset vector located at 0x00 8000 at the beginning of program memory. The Reset initialization routine is programmed in ROM by STMicroelectronics.
- Fixed interrupt vector addresses located at the high addresses of the memory map (0x00 8004 to 0x00 807C) sorted by hardware priority order.

5.2 Interrupt masking and processing flow

The interrupt masking is managed by bits I1 and I0 of the CCR register and by the ITC_SPRx registers which set the software priority level of each interrupt vector (see *Table 8*). The processing flow is shown in *Figure 6*.

When an interrupt request has to be serviced:

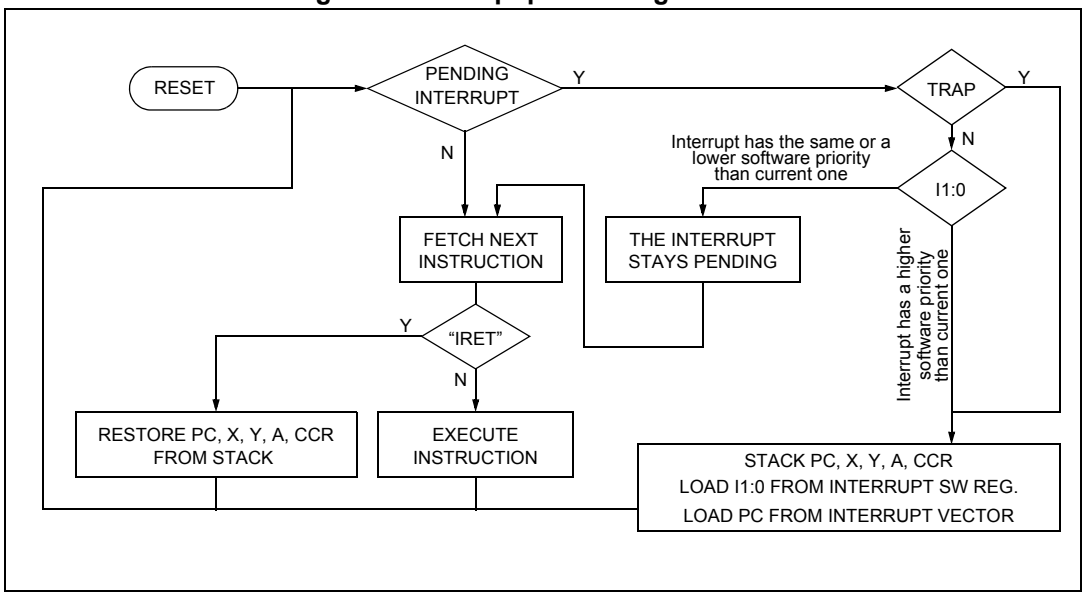
1. Normal processing is suspended at the end of the current instruction execution.
2. The PC, X,Y, A and CCR registers are saved onto the stack.
3. Bits I1 and I0 of CCR register are set according to the values in the ITC_SPRx registers corresponding to the serviced interrupt vector.
4. The PC is then loaded with the interrupt vector of the interrupt to service and the first instruction of the interrupt service routine is fetched (refer to the datasheet interrupt mapping table for details on vector addresses).

The interrupt service routine should end with the IRET instruction which causes the content of the saved registers to be recovered from the stack. As a consequence of the IRET instruction, bits I1 and I0 are restored from the stack and the program execution resumes.

Table 8. Software priority levels

Software priority	Level	I1	I0
Level 0 (main)	Low ↓ High	1	0
Level 1		0	1
Level 2		0	0
Level 3 (= software priority disabled)		1	1

Figure 6. Interrupt processing flowchart



Caution: If the interrupt mask bits I0 and I1 are set within an interrupt service routine (ISR) with the instruction SIM, removal of the interrupt mask with RIM causes the software priority to be set to level 0.

To restore the correct priority when disabling and enabling interrupts inside an ISR, follow the procedures presented in *Table 8* for disabling and enabling the interrupts.

Table 9. Interrupt enabling/disabling inside an ISR

Disabling the interrupts	Enabling the interrupts
#asm PUSH CC POP ISR_CC(1) SIM #endasm	#asm PUSH ISR_CC(1) POP CC #endasm

1. IRS_CC is a variable which stores the current value of the CC register.

5.2.1 Servicing pending interrupts

Several interrupts can be pending at the same time. The interrupt to be taken into account is determined by the following two-step process:

1. The highest software priority interrupt is serviced.
2. If several interrupts have the same software priority then the interrupt with the highest hardware priority is serviced first.

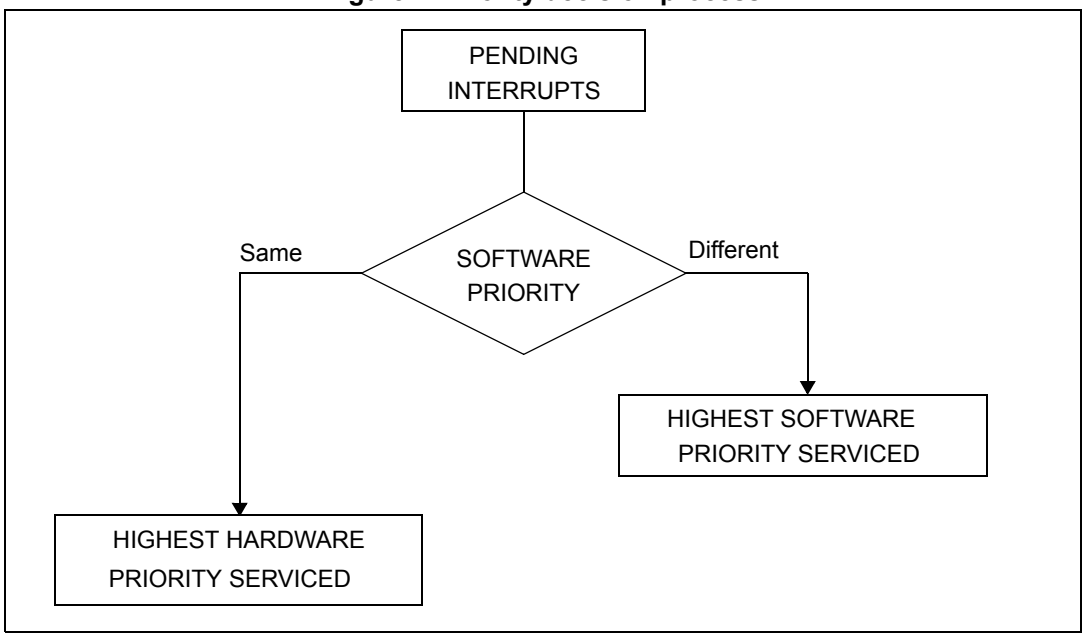
When an interrupt request is not serviced immediately, it is latched and then processed when its software priority combined with the hardware priority becomes the highest one.

Note: 1 The hardware priority is exclusive while the software one is not. This allows the previous process to succeed with only one interrupt.

2 RESET and TRAP are considered as having the highest software priority in the decision process.

See *Figure 7* for a description of pending interrupt servicing process.

Figure 7. Priority decision process



5.2.2 Interrupt sources

Two interrupt source types are managed by the STM8 interrupt controller:

- Non-maskable interrupts: RESET and TRAP
- Maskable interrupts: external interrupts or interrupts issued by internal peripherals

Non-maskable interrupt sources

Non-maskable interrupt sources are processed regardless of the state of bits I1 and I0 of the CCR register (see *Figure 6*). PC, X, Y, A and CCR registers are stacked only when a TRAP interrupt occurs. The corresponding vector is then loaded in the PC register and bits I1 and I0 of the CCR register are set to disable interrupts (level 3).

- TRAP (non-maskable software interrupt)

  This software interrupt source is serviced when the TRAP instruction is executed. It is serviced according to the flowchart shown in *Figure 6*.

  A TRAP interrupt does not allow the processor to exit from Halt mode.
- RESET

  The RESET interrupt source has the highest STM8 software and hardware priorities. This means that all the interrupts are disabled at the beginning of the reset routine. They must be re-enabled by the RIM instruction (see *Table 12: Dedicated interrupt instruction set*).

  A RESET interrupt allows the processor to exit from Halt mode.

See RESET chapter for more details on RESET interrupt management.

Maskable interrupt sources

Maskable interrupt vector sources are serviced if the corresponding interrupt is enabled and if its own interrupt software priority in ITC_SPRx registers is higher than the one currently being serviced (I1 and I0 in CCR register). If one of these two conditions is not met, the interrupt is latched and remains pending.

- External interrupts

  External interrupts can be used to wake up the MCU from Halt mode. The device sensitivity to external interrupts can be selected by software through the External Interrupt Control registers (EXTI_CRx).

  When several input pins connected to the same interrupt line are selected simultaneously, they are logically ORed.

  When external level-triggered interrupts are latched, if the given level is still present at the end of the interrupt routine, the interrupt remains activated except if it has been inactivated in the routine.

- Peripheral interrupts

  peripheral interrupts cause the MCU to wake up from Halt mode. See the interrupt vector table in the datasheet.

  A peripheral interrupt occurs when a specific flag is set in the peripheral status register and the corresponding enable bit is set in the peripheral control register.

  The standard sequence for clearing a peripheral interrupt performs an access to the status register followed by a read or write to an associated register. The clearing sequence resets the internal latch. A pending interrupt (that is an interrupt waiting to be serviced) is therefore lost when the clear sequence is executed.

5.3 Interrupts and low power modes

All interrupts allow the processor to exit from Wait mode.

Only a Reset or an event allows the processor to exit from Low power wait mode. This mode is entered by executing a WFE instruction in Low power run mode. The wakeup by an event makes the system go back to Low power run mode (refer to *Section 9: Power management* for more details).

Only external and other specific interrupts allow the processor to exit from Halt and Active-halt mode (see wakeup from halt and wakeup from Active-halt).

When several pending interrupts are present while waking up from Halt mode, the first interrupt serviced can only be an interrupt with exit-from-Halt mode capability. It is selected through the decision process shown in *Figure 7*. If the highest priority pending interrupt cannot wake up the device from Halt mode, it will be serviced next.

If any internal or external interrupt (from a timer for example) occurs while the HALT instruction is executing, the HALT instruction is completed but the interrupt invokes the wakeup process immediately after the HALT instruction has finished executing. In this case the MCU is actually waking up from Halt mode to Run mode, with the corresponding delay of t_{WUH} as specified in the datasheet.

5.4 Activation level/low power mode control

The MCU activation level is configured by programming the AL bit in the CFG_GCR register (see global configuration register (CFG_GCR)).

This bit is used to control the low power modes of the MCU. In very low power applications, the MCU spends most of the time in WFI/Halt mode and is woken up (through interrupts) at specific moments in order to execute a specific task. Some of these recurring tasks are short enough to be treated directly in an ISR (interrupt service routine), rather than going back to the main program. To cover this case, you can set the AL bit before entering Low power mode (by executing WFI/HALT instruction), then the interrupt routine returns directly to Low power mode. The run time/ISR execution is reduced due to the fact that the register context is saved only on the first interrupt.

As a consequence, all the operations can be executed in ISR in very simple applications. In more complex ones, an interrupt routine may relaunch the main program by simply resetting the AL bit.

For example, an application may need to be woken up by the auto-wakeup unit (AWU) every 50 ms in order to check the status of some pins/sensors/push-buttons. Most of the time, as these pins are not active, the MCU can return to Low power mode without running the main program. If one of these pins is active, the ISR decides to launch the main program by resetting the AL bit.

5.5 Concurrent and nested interrupt management

STM8 devices feature two interrupt management modes:

- Concurrent mode
- Nested mode

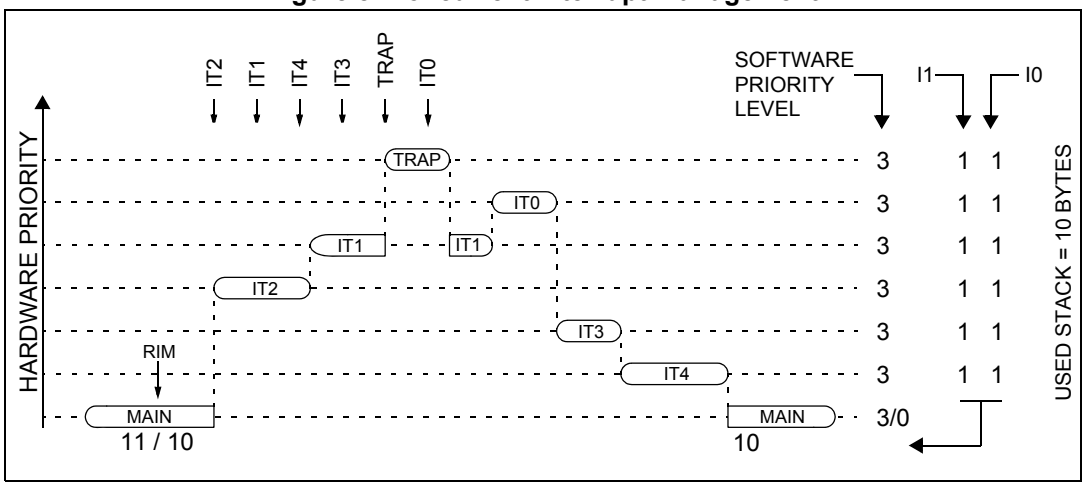
5.5.1 Concurrent interrupt management mode

In this mode, all interrupts are interrupt priority level 3 so that none of them can be interrupted, except by a RESET, or TRAP.

The hardware priority is given in the following order from the lowest to the highest priority, that is: MAIN, IT4, IT3, IT2, IT1, IT0, TRAP (same priority), and RESET.

Figure 8 shows an example of concurrent interrupt management mode.

Figure 8. Concurrent interrupt management



5.5.2 Nested interrupt management mode

In this mode, interrupts are allowed during interrupt routines. This mode is activated as soon as an interrupt priority level lower than level 3 is set.

The hardware priority is given in the following order from the lowest to the highest priority, that is: MAIN, IT4, IT3, IT2, IT1, IT0, and TRAP.

The software priority is configured for each interrupt vector by setting the corresponding I1_x and I0_x bits of the ITC_SPRx register. I1_x and I0_x bits have the same meaning as I1 and I0 bits of the CCR register (see *Table 10*).

Level 0 can not be programmed (I1_x=1, I0_x=0). In this case, the previously stored value is kept. For example: if previous value is 0xCF, and programmed value equals 64h, the result is 44h.

The RESET and TRAP vectors have no software priorities. When one is serviced, bits I1 and I0 of the CCR register are both set.

Caution: If bits I1_x and I0_x are modified while the interrupt x is executed, the device operates as follows: if the interrupt x is still pending (new interrupt or flag not cleared) and the new software priority is higher than the previous one, then the interrupt x is re-entered. Otherwise, the software priority remains unchanged till the next interrupt request (after the IRET of the interrupt x).

During the execution of an interrupt routine, the HALT, POPCC, RIM, SIM and WFI instructions change the current software priority till the next IRET instruction or one of the previously mentioned instructions is issued. See *Section 5.7* for the list of dedicated interrupt instructions.

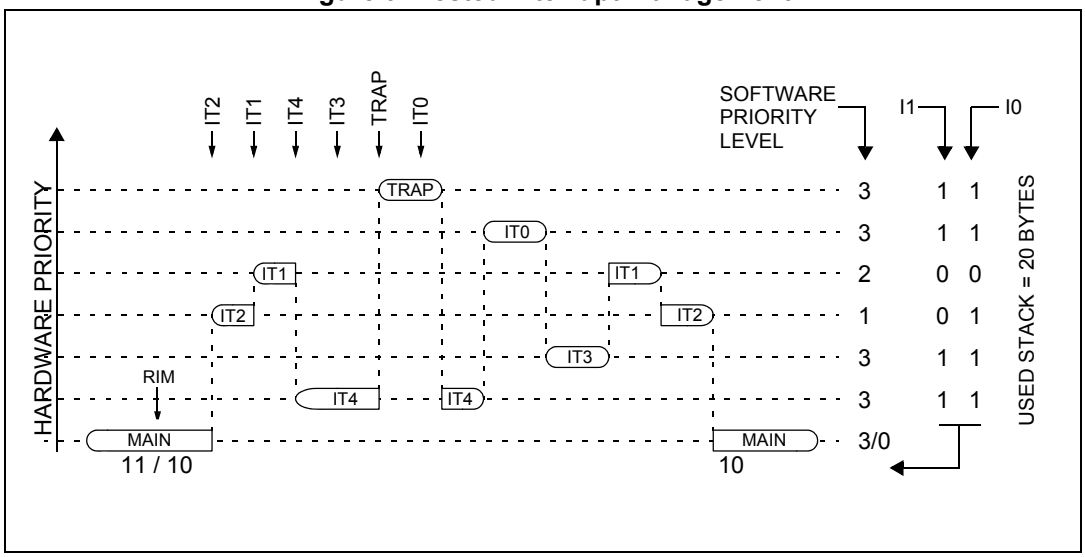
Figure 9 shows an example of nested interrupt management mode.

Warning: A stack overflow may occur without notifying the software of the failure.

Table 10. Vector address map versus software priority bits

Vector address	ITC_SPRx bits
0x00 8008h	I1_0 and I0_0 bits
0x00 800Ch	I1_1 and I0_1 bits
...	...
0x00 807Ch	I1_29 and I0_29 bits

Figure 9. Nested interrupt management



5.6 External interrupts

Ten interrupt vectors are dedicated to external Interrupt events:

- EXTIB - 8 lines on Port B: PB[7:0]
- EXTID - 8 lines on Port D: PD[7:0]
- EXTI0 - 4 lines on Port A/B/C/D, bit 0: Px[0]
- EXTI1 - 3 lines on Port A/B/D
- EXTI2 - 3 lines on Port A/B/D
- EXTI3 - 3 lines on Port A/B/D
- EXTI4 - 3 lines on Port A/B/D
- EXTI5 - 2 lines on Port B/D
- EXTI6 - 3 lines on Port A/B/D
- EXTI7 - 3 lines on Port A/B/D

To generate an interrupt, the corresponding GPIO port must be configured in input mode with interrupts enabled. Refer to the register description in the GPIO chapter for details.

When an external interrupt occurs, the corresponding bit is set in the EXTI_SRx status register. This indicates a pending interrupt. Clearing this bit, writing a 1 in it, clears the corresponding pending external interrupt.

The interrupt sensitivity must be configured in the external interrupt control register 1 (EXTI_CR1), external interrupt control register 2 (EXTI_CR2), and external interrupt control register 3 (EXTI_CR3) (see *Section 5.9.5*, *Section 5.9.6* and *Section 5.9.7*).

Table 11. External interrupt sensitivity

GPIO port	Interrupt sensitivity	Configuration register
EXTI0 to EXTI3 on port A, B, C and D	Falling edge and low level Rising edge only Falling edge only Rising and falling edge	EXTI_CR1
EXTI4 to EXTI7 of port A, B, C and D		EXTI_CR2
EXTIB and EXTID		EXTI_CR3

5.7 Interrupt instructions

Table 12 shows the interrupt instructions.

Table 12. Dedicated interrupt instruction set

Instruction	New description	Function/example	I1	H	I0	N	Z	C
HALT	Entering Halt mode		1		0			
IRET	Interrupt routine return	Pop CCR, A, X, Y, PC	I1	H	I0	N	Z	C
JRM	Jump if I1:0=11 (level 3)	I1:0=11 ?						
JRNM	Jump if I1:0<>11	I1:0<>11 ?						
POP CC	Pop CCR from the stack	Memory => CCR	I1	H	I0	N	Z	C
PUSH CC	Push CC on the stack	CC =>Memory						

Table 12. Dedicated interrupt instruction set (continued)

Instruction	New description	Function/example	I1	H	I0	N	Z	C
RIM	Enable interrupt (level 0 set)	Load 10 in I1:0 of CCR	1		0			
SIM	Disable interrupt (level 3 set)	Load 11 in I1:0 of CCR	1		1			
TRAP	Software trap	Software NMI	1		1			
WFI	Wait for interrupt		1		0			
WFE	Wait for event		1		0			

5.8 Interrupt mapping

Refer to the corresponding device datasheet for the table of interrupt vector addresses.

5.9 ITC and EXTI registers

5.9.1 CPU condition code register interrupt bits (CCR)

Address: refer to the general hardware register map table in the datasheet.
Reset value: 0x28

7	6	5	4	3	2	1	0
V	–	I1	H	I0	N	Z	C
r	r	rw	r	rw	r	r	r

Bits 5, 3[1] **I[1:0]:** Software interrupt priority bits[2]

These two bits indicate the software priority of the current interrupt request. When an interrupt request occurs, the software priority of the corresponding vector is loaded automatically from the software priority registers (ITC_SPRx).

The I[1:0] bits can be also set/cleared by software using the RIM, SIM, HALT, WFI, IRET or PUSH/POP instructions (see *Figure 9: Nested interrupt management*).

I1	I0	Priority	Level
1	0	Level 0 (main)	Low ↓ High
0	1	Level 1	
0	0	Level 2	
1	1	Level 3 (= software priority disabled*)	

1. Refer to the central processing section for details on the other CCR bits.
2. TRAP and RESET events can interrupt a level-3 program.

5.9.2 Software priority register x (ITC_SPRx)

Address offset: 0x00 to 0x07
Reset value: 0xFF

	7	6	5	4	3	2	1	0
ITC_SPR1	Reserved		VECT2SPR[1:0]		VECT1SPR[1:0]		Reserved	
ITC_SPR2	VECT7SPR[1:0]		VECT6SPR[1:0]		Reserved		VECT4SPR[1:0]	
ITC_SPR3	VECT11SPR[1:0]		VECT10SPR[1:0]		VECT9SPR[1:0]		VECT8SPR[1:0]	
ITC_SPR4	VECT15SPR[1:0]		VECT14SPR[1:0]		VECT13SPR[1:0]		VECT12SPR[1:0]	
ITC_SPR5	VECT19SPR[1:0]		Reserved		Reserved		Reserved	
ITC_SPR6	Reserved		VECT22SPR[1:0]		VECT21SPR[1:0]		VECT20SPR[1:0]	
ITC_SPR7	VECT27SPR[1:0]		VECT26SPR[1:0]		VECT25SPR[1:0]		Reserved	
ITC_SPR8	Reserved				VECT29SPR[1:0]		VECT28SPR[1:0]	
	rw				rw	rw	rw	rw

Bits 7:0 **VECTxSPR[1:0]:** Vector x software priority bits

These eight read/write registers (ITC_SPR1 to ITC_SPR8) are written by software to define the software priority of each interrupt vector.

The list of vectors is given in *Table 10: Vector address map versus software priority bits*.

Refer to *Section 5.9.1: CPU condition code register interrupt bits (CCR)* for the values to be programmed for each priority.

Reserved

ITC_SPR8 bits 7:4 are forced to 1 by hardware.

Note: It is forbidden to write 10 (priority level 0). If 10 is written, the previous value is kept and the interrupt priority remains unchanged.

5.9.3 External interrupt control register 1 (EXTI_CR1)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
P3IS[1:0]		P2IS[1:0]		P1IS[1:0]		P0IS[1:0]	
rw		rw		rw		rw	

Bits 7:6 **P3IS[1:0]:** Portx bit 3 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 3 of Port A, B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 5:4 **P2IS[1:0]:** Portx bit 2 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 2 of Port A, B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 3:2 **P1IS[1:0]:** Portx bit 1 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 1 of Port A, B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 1:0 **P0IS[1:0]:** Portx bit 0 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of bit 0 of Port A, B, C, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

5.9.4 External interrupt control register 2 (EXTI_CR2)

Address offset: 0x01
Reset value: 0x00

7	6	5	4	3	2	1	0
P7IS[1:0]		P6IS[1:0]		P5IS[1:0]		P4IS[1:0]	
rw		rw		rw		rw	

Bits 7:6 **P7IS[1:0]:** Portx bit 7 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 7 of Port A, B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 5:4 **P6IS[1:0]:** Portx bit 6 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 6 of Port A, B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 3:2 **P5IS[1:0]:** Portx bit 5 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 5 of Port B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 1:0 **P4IS[1:0]:** Portx bit 4 external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the bit 4 of Port A, B, and/or D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

5.9.5 External interrupt control register 3 (EXTI_CR3)

Address offset: 0x02
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				PDIS[1:0]		PBIS[1:0]	
				rw		rw	

Bits 7:4 Reserved

Bits 3:2 **PDIS[1:0]:** Port D external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the Port D external interrupts, when EXTID for Port D[3:0] and/or Port D[7:4] is enabled.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Bits 1:0 **PBIS[1:0]:** Port B external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of the Port B external interrupts, when EXTIB for Port B[3:0] and/or PortB[7:4] is enabled.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

5.9.6 External interrupt status register 1 (EXTI_SR1)

Address offset: 0x03
Reset value: 0x00

7	6	5	4	3	2	1	0
P7F	P6F	P5F	P4F	P3F	P2F	P1F	P0F
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 7:0 **PxF:** Port A/B/C/D bit x external interrupt flag

These bits are set by hardware when an interrupt event occurs on the corresponding pin.They are cleared by writing a ‘1’ by software.

0: No interrupt
1: External interrupt pending

5.9.7 External interrupt status register 2 (EXTI_SR2)

Address offset: 0x04
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						PDF	PBF
						rc_w1	rc_w1

Bits 7:2 Reserved

Bit 1 **PDF:** Port D external interrupt flag
This bit is set by hardware when an interrupt event occurs on the corresponding pin. It is cleared by writing a ‘1’ by software.
0: No interrupt
1: External interrupt pending

Bit 0 **PBF:** Port B external interrupt flag
This bit is set by hardware when an interrupt event occurs on the corresponding pin.It is cleared by writing a ‘1’ by software.
0: No interrupt
1: External interrupt pending

5.9.8 External interrupt port select register (EXTI_CONF)

Address offset: 0x05
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				PDHIS	PDLIS	PBHIS	PBLIS
				rw	rw	rw	rw

Bits 7:4 Reserved

Bit 3 **PDHIS:** Port D[7:4] external interrupt select
It selects pins PD[7:4] for EXTID interrupt.
0: PD[7:4] are used for EXTI7-EXTI4 interrupt generation
1: PD[7:4] are used for EXTID interrupt generation

Bit 2 **PDLIS:** Port D[3:0] external interrupt select
It selects pins PD[3:0] for EXTID interrupt.
0: PD[3:0] are used for EXTI3-EXTI0 interrupt generation
1: PD[3:0] are used for EXTID interrupt generation

Bit 1 **PBHIS:** Port B[7:4] external interrupt select
It selects pins PB[7:4] for EXTIB interrupt.
0: PB[7:4] are used for EXTI7-EXTI4 interrupt generation
1: PB[7:4] are used for EXTIB interrupt generation

Bit 0 **PBLIS:** Port B[3:0] external interrupt select
It selects pins PB[3:0] for EXTIB interrupt.
0: PB[3:0] are used for EXTI3-EXTI0 interrupt generation
1: PB[3:0] are used for EXTIB interrupt generation

5.9.9 ITC and EXTI register map and reset values

Table 13. ITC and EXTI register map

Add. offset	Register name	7	6	5	4	3	2	1	0
ITC-SPR block(1)									
0x00	ITC_SPR1 Reset value	- 1	- 1	VECT2 SPR1 1	VECT2 SPR0 1	VECT1 SPR1 1	VECT1 SPR0 1	- 1	- 1
0x01	ITC_SPR2 Reset value	VECT7 SPR1 1	VECT7 SPR0 1	VECT6 SPR1 1	VECT6 SPR0 1	- 1	- 1	VECT4 SPR1 1	VECT4 SPR0 1
0x02	ITC_SPR3 Reset value	VECT11 SPR1 1	VECT11 SPR0 1	VECT10 SPR1 1	VECT10 SPR0 1	VECT9 SPR1 1	VECT9 SPR0 1	VECT8 SPR1 1	VECT8 SPR0 1
0x03	ITC_SPR4 Reset value	VECT15 SPR1 1	VECT15 SPR0 1	VECT14 SPR1 1	VECT14 SPR0 1	VECT13 SPR1 1	VECT13 SPR0 1	VECT12 SPR1 1	VECT12 SPR0 1
0x04	ITC_SPR5 Reset value	VECT19 SPR1 1	VECT19 SPR0 1	- 1	- 1	- 1	- 1	- 1	- 1
0x05	ITC_SPR6 Reset value	- 1	- 1	VECT22 SPR1 1	VECT22 SPR0 1	VECT21 SPR1 1	VECT21 SPR0 1	VECT20 SPR1 1	VECT20 SPR0 1
0x06	ITC_SPR7 Reset value	VECT27 SPR1 1	VECT27 SPR0 1	VECT26 SPR1 1	VECT26 SPR0 1	VECT25 SPR1 1	VECT25 SPR0 1	- 1	- 1
0x07	ITC_SPR8 Reset value	- 1	- 1	- 1	- 1	VECT29 SPR1 1	VECT29 SPR0 1	VECT28 SPR1 1	VECT28 SPR0 1
ITC-EXTI block(2)									
0x00	EXTI_CR1 Reset value	P3IS1 0	P3IS0 0	P2IS1 0	P2IS0 0	P1IS1 0	P1IS0 0	P0IS1 0	P0IS0 0
0x01	EXTI_CR2 Reset value	P7IS1 0	P7IS0 0	P6IS1 0	P6IS0 0	P5IS1 0	P5IS0 0	P4IS1 0	P4IS0 0
0x02	EXTI_CR3 Reset value	- 0	- 0	- 0	- 0	PDIS1 0	PDIS0 0	PBIS1 0	PBIS0 0
0x03	EXTI_SR1 Reset value	P7F 0	P6F 0	P5F 0	P4F 0	P3F 0	P2F 0	P1F 0	P0F 0
0x04	EXTI_SR2 Reset value	- 0	- 0	- 0	- 0	- 0	- 0	PDF 0	PBF 0
0x05	EXTI_CONF Reset value	- 0	- 0	- 0	- 0	PDHIS 0	PDLIS 0	PBHIS 0	PBLIS 0

1. The address offsets are expressed for the ITC-SPR block base address (see Table CPU/SWIM/debug module/interrupt controller registers in the datasheet).
2. The address offsets are expressed for the ITC-EXTI block base address (see Table General hardware register map in the datasheet).

6 Power supply

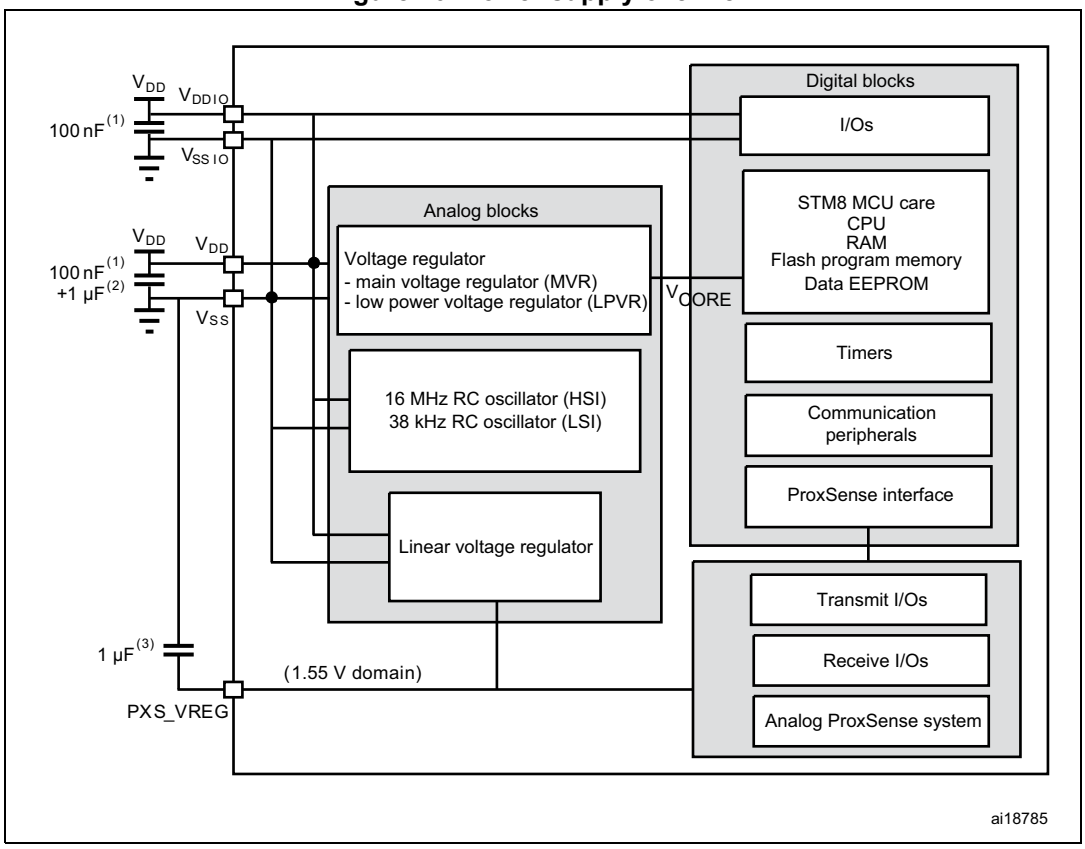
The MCU has one power supply, V_{DD}/V_{SS} (1.65 to 3.6 V), to supply both the I/Os and the internal voltage regulator. The voltage regulator has 2 modes: Main Voltage Regulator mode (MVR) and Low Power Voltage Regulator mode (LPVR).

When entering Halt or Active-halt modes (described in *Section 9.4.2: Halt mode* and *Section 9.4.3: Active-halt mode*), the system automatically switches from the MVR to the LPVR in order to reduce current consumption.

The power-on reset (POR) guarantees a safe reset when the MCU is powered on.

Refer to the electrical characteristics section in the datasheet for details on the operating range.

Figure 10. Power supply overview



1. Each power supply pair must be decoupled with filtering ceramic capacitors as shown above. These capacitors must be placed as close as possible to, or below, the appropriate pins to ensure the correct functionality of the device.
2. This 1 µF capacitor must be connected to the V_{DD} pin.
3. This 1 µF ceramic capacitor connected to PXS_VREG must be low ESR (ESR $\leq 1\Omega$).

Note: The different power modes are described in Section 9: Power management on page 67.

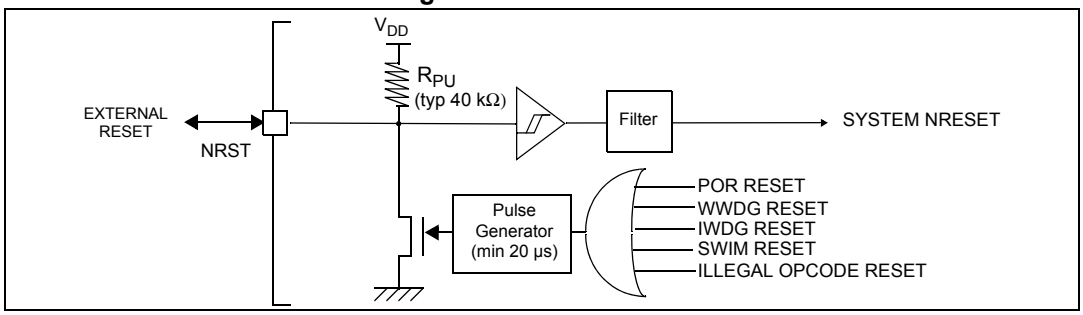
7 Reset (RST) and voltage detection

There are six reset sources:

- External reset through the NRST pin (this pin can also be configured as general purpose output)
- Power-on reset (POR)
- Independent watchdog reset (IWDG)
- Window watchdog (WWDG)
- SWIM reset
- Illegal opcode reset

These sources act on the $\overline{\text{RESET}}$ pin. The RESET service routine vector is fixed at address 0x8000 in the memory map.

Figure 11. Reset circuit



7.1 “Reset state” and “under reset” definitions

When a reset occurs, there is a reset phase from the external pin pull-down to the internal reset signal release. During this phase, the microcontroller sets some hardware configurations before going to the reset vector.

At the end of this phase, most of the registers are configured with their “reset state” values. During the reset phase, i.e. “under reset”, some pin configurations may be different from their “reset state” configuration.

7.2 External reset (NRST pin)

7.2.1 Asynchronous external reset description

The NRST pin is both an input and an open-drain output with integrated R_{PU} weak pull-up resistor.

A minimum of 300 ns low pulse on the NRST pin is needed to generate an external reset. The reset detection is asynchronous and therefore the MCU can enter reset even in Halt mode.

The NRST pin also acts as an open-drain output for resetting external devices.

An internal temporization maintains a pulse of at least 20 µs for every internal reset source. An additional internal weak pull-up ensures a high level on the reset pin when the reset is not forced.

Refer to *Figure 11* and see electrical parameters section in the datasheet for more details.

7.2.2 Configuring NRST/PA5 pin as general purpose output

To optimize the number of available pins, the NRST pin (external reset) can be configured as a general purpose push-pull output (PA5).

For security, this configuration can be performed once only after reset, by writing a specified key (D0h) to the *Reset pin configuration register (RST_CR)*.

When the PA5 pin is configured as a general purpose output the MCU can be reset only by a power-on reset (POR) or other internal reset source.

7.3 Internal reset

For internal reset sources, the $\overline{\text{RESET}}$ pin is kept low during the delay phase generated by the pulse generator.

Each internal reset source is linked to a specific flag bit in the *Reset status register (RST_SR)*. These flags are set respectively at reset state depending on the given reset source. So they are used to identify the last reset source. They are cleared by software writing the logic value “1”.

Note: All flags besides the POR flag are reset at POR.

7.3.1 Power-on reset (POR)

During power-on, the POR keeps the device under reset until the supply voltage (V_{DD}) reaches a specified voltage and then holds reset active even longer for a specified time in order to assure that V_{DD} has reached the minimum operating voltage before releasing the $\overline{\text{RESET}}$ pin. See electrical parameters section in the datasheet for more details.

7.3.2 Independent watchdog reset

See the independent watchdog section for details.

A reset can be triggered by the application software using the independent watchdog.

7.3.3 SWIM reset

An external device connected to the SWIM interface can request the SWIM block to generate an MCU reset.

7.3.4 Illegal opcode reset

In order to provide enhanced robustness to the device against unexpected behavior, a system of illegal opcode detection is implemented. If a code to be executed does not correspond to any opcode or prebyte value, a reset is generated. This, combined with the watchdog, allows recovery from an unexpected fault or interference.

7.4 RST registers

7.4.1 Reset pin configuration register (RST_CR)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
RSTPIN_KEY							
rwo							

Bits 7:0 **RSTPIN_KEY[7:0]**: Reset pin configuration key
0x00: NRST/PA5 configured as reset pin (default reset value)
0xD0: NRST/PA5 configured as general purpose output
These bits are write once only. They can also be read at any time.
Note: Writing any value beside D0h is equivalent to writing 00h.

7.4.2 Reset status register (RST_SR)

Address offset: 0x01
Reset value after power-on reset: 0x01

7	6	5	4	3	2	1	0
Reserved			WWDGF	SWIMF	ILLOPF	IWDGF	PORF
			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 7:5 Reserved

Bit 4 **WWDGF**: Window Watchdog reset flag
0: No WWDG reset occurred
1: A WWDG reset occurred
This bit is set by hardware and cleared by software writing “1”.

Bit 3 **SWIMF**: SWIM reset flag
0: No SWIM reset occurred
1: A SWIM reset occurred
This bit is set by hardware and cleared by software writing “1”.

Bit 2 **ILLOPF**: Illegal opcode reset flag
0: No ILLOP reset occurred
1: An ILLOP reset occurred
This bit is set by hardware and cleared by software writing “1”.

Bit 1 **IWDGF**: Independent Watchdog reset flag
0: No IWDG reset occurred
1: An IWDG reset occurred
This bit is set by hardware and cleared by software writing “1”.

Bit 0 **PORF**: Power-on reset (POR) flag
0: No POR occurred
1: A POR occurred
This bit is set by hardware and cleared by software writing “1”.

Note: RST_SR reset value depends on the previous reset source.

7.5 RST register map and reset values

Table 14. RST register map and reset values

Address offset(1)	Register name	7	6	5	4	3	2	1	0
0x00	RST_CR Reset value	RSTPIN_KEY7 0	RSTPIN_KEY6 0	RSTPIN_KEY5 0	RSTPIN_KEY4 0	RSTPIN_KEY3 0	RSTPIN_KEY2 0	RSTPIN_KEY1 0	RSTPIN_KEY0 0
0x01	RST_SR Reset value	- 0	- 0	- 0	WWDGF 0	SWIMF 0	ILLOPF 0	IWDGF 0	PORF 1

1. Please refer to the “general hardware register map” table in the datasheet for details on register addresses.

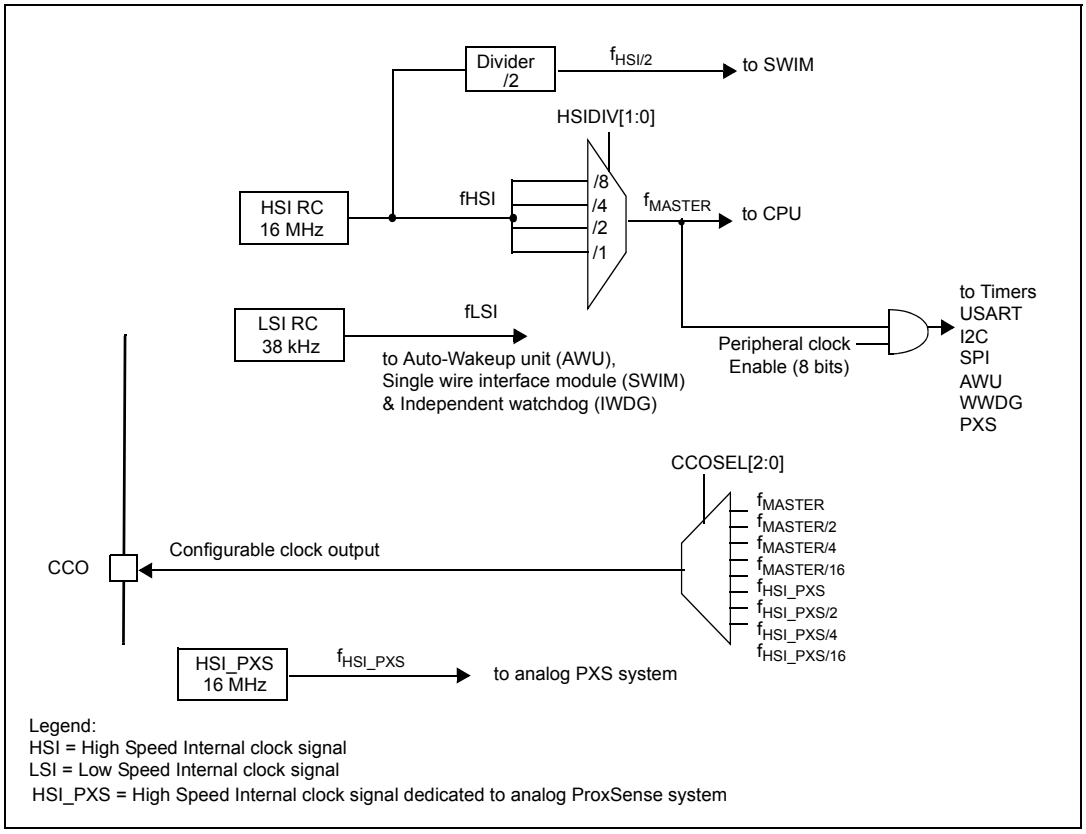
8 Clock control (CLK)

The clock controller is designed to be very robust and at the same time easy to use. Its purpose is to obtain the best performance in the application while at the same time get the full benefit of all the microcontroller power saving capabilities.

You can manage the clock distribution to the CPU and to the various peripherals, in order to optimize the power consumption.

A safe and glitch-free switch mechanism allows you to switch the master clock division factor on the fly, by means of clock prescaler.

Figure 12. Clock structure



8.1 Master clock (HSI clock)

The master clock of the system is a 16-MHz high-speed internal RC oscillator (HSI RC), followed by a programmable prescaler.

The f_{HSI} clock signal generated from the internal 16-MHz RC oscillator can then be divided by a programmable divider (factor 1 to 8) to obtain f_{MASTER}. This is programmed in the *Clock divider register (CLK_CKDIVR)*.

Note: At startup the master clock source is automatically selected as HSI RC clock output divided by 8 (HSI/8).

The HSI RC oscillator has the advantage of providing a 16-MHz master clock source with 50% duty cycle at low cost (no external components) and with a fast startup time.

The clock controller configures the master clock source as HSI RC clock output divided by 8 ($f_{HSI}/8$), which helps ensure a safe startup in case of poor V_{DD} conditions.

8.1.1 Peripheral clock gating (PCG)

Gating the clock to unused peripherals helps reduce power consumption. Peripheral clock gating (PCG) mode selectively enables or disables the f_{MASTER} clock connection to the following peripherals at any time in run or slow mode:

- TIM2
- TIM3
- TIM4
- I2C
- SPI
- USART
- AWU/BEEP (except LSI clock controlled by AWUEN bit in the *Control/status register (AWU_CSR)* and by BEEPEN bit in the *Beeper control/status register (BEEP_CSR)*).
- PXS (ProxSense)
- WWDG

  The Window Watchdog (WWDG) peripheral clock is the only one to be enabled by default as it can be started by hardware if the corresponding option bit is set.

After a device reset, all peripheral clocks are disabled, except the WWDG clock. You can enable the clock to any peripheral by setting the corresponding PCKEN bit in the *Peripheral clock gating register 1 (CLK_PCKENR1)* or *Peripheral clock gating register 2 (CLK_PCKENR2)*. To enable a peripheral, first enable the corresponding PCKEN bit in the CLK_PCKENR registers and then set the peripheral enable bit in the peripheral control registers.

To disable properly the peripheral, first disable the appropriate bit in the peripheral control registers and then stop the corresponding clock.

The AWU counter is fed by the low speed internal specific clock (f_{LSI}) different from f_{MASTER}, so that it continues to run even if the clock gating to this peripheral register is asserted. The same is true for the beeper.

8.2 LSI clock

The LSI RC oscillator generates the f_{LSI} clock. This clock is a 38-kHz Low Speed Internal RC oscillator (LSI RC). It is used by the independent watchdog (IWDG), Auto-Wakeup unit (AWU) and beeper (BEEP). The LSI is active in Active-halt mode and can be kept in run mode.

8.3 Configurable clock-output capability (CCO)

The configurable clock output (CCO) capability outputs a clock on the external CLK_CCO pin. The master clock is used to drive the clock output with a programmable prescaler which can divide the clock frequency by a factor of 1, 2, 4 or 16.

The selection is controlled by the CCOSEL[2:0] bits in the *Configurable clock output register (CLK_CCOR)*.

The sequence to output the targeted clock starts with CCOEN=1 in *Configurable clock output register (CLK_CCOR)*. When CCOEN=1, the CCOSEL[2:0] bits are write protected.

To disable the clock output the user has to clear the CCOEN bit.

The configurable clock output has a configurable slope which must be adapted according to the output frequency. It is recommended to select a buffer slope at least twice that of the targeted output frequency.

8.4 CLK registers

8.4.1 Clock divider register (CLK_CKDIVR)

Address offset: 0x00
Reset value: 0x03

7	6	5	4	3	2	1	0
Reserved						HSIDIV[1:0]	
						rw	

Bits 7:2 Reserved

Bits 1:0 **HSIDIV[1:0]**: High speed internal clock prescaler

00: $f_{MASTER} = f_{HSI\ RC\ output}$
01: $f_{MASTER} = f_{HSI\ RC\ output}/2$
10: $f_{MASTER} = f_{HSI\ RC\ output}/4$
11: $f_{MASTER} = f_{HSI\ RC\ output}/8$

These bits are written by software to define the HSIDIV prescaling factor.

8.4.2 Peripheral clock gating register 1 (CLK_PCKENR1)

Address offset: 0x03
Reset value: 0x00

7	6	5	4	3	2	1	0
PCKENR1[7:0]							
rw							

Bits 7:0 **PCKENR1[7:0]**: Peripheral clock enable

0: f_{MASTER} to peripheral disabled
1: f_{MASTER} to peripheral enabled

These bits are written by software to enable or disable the f_{MASTER} clock to the corresponding peripheral. See *Table 15*

Table 15. Peripheral clock gating bits

Control bit	Peripheral
PCKENR17	ProxSense
PCKENR16	AWU (except LSI clock controlled by AWUEN bit in the *Control/status register (AWU_CSR)*).
PCKENR15	USART
PCKENR14	SPI
PCKENR13	I^2C
PCKENR12	TIM4
PCKENR11	TIM3
PCKENR10	TIM2

8.4.3 Peripheral clock gating register 2 (CLK_PCKENR2)

Address offset: 0x04
Reset value: 0x01

7	6	5	4	3	2	1	0
Reserved							PCKENR20
							rw

Bits 7:1 Reserved

Bit 0 **PCKENR20**: WWDG peripheral clock enable

0: f_{MASTER} to peripheral disabled
1: f_{MASTER} to peripheral enabled (reset value)

8.4.4 Configurable clock output register (CLK_CCOR)

Address offset: 0x05
Reset value: 0x10

7	6	5	4	3	2	1	0
Reserved		CCOSLP[1:0]		CCOSEL[2:0]			CCOEN
		rw		rw			rw

Bits 7:6 Reserved

Bits 5:4 **CCOSLP[1:0]**: Configurable clock output buffer slope.

00: 400 kHz
01: 20 MHz
10: 10 MHz
11: 50 MHz

Bits 3:1 **CCOSEL[2:0]**: Configurable clock output selection.

These bits are written by software to select the source of the output clock available on the CLK_CCO pin.

000: f_{MASTER}
001: $f_{MASTER}/2$
010: $f_{MASTER}/4$
011: $f_{MASTER}/16$
100: f_{PXS}
101: $f_{PXS}/2$
110: $f_{PXS}/4$
111: $f_{PXS}/16$

Note: To correctly output the ProxSense RC signal, the following sequence must be respected:

– Enable the PXS clock peripheral by setting bit PCKENR17 in the CLK_PCKENR1 register.
– Configure CCOSEL[2:0] and set bit CCO in the CLK_CCOR register.
– Enable the ProxSense peripheral by setting bit PXS_EN in the PXS_CR1 register.

Bit 0 **CCOEN**: Configurable clock output enable

This bit is set and cleared by software.

0: CCO clock output disabled
1: CCO clock output enabled

8.4.5 CLK register map and reset values

Table 16. CLK register map and reset values

Address offset (1)	Register name	7	6	5	4	3	2	1	0
0x00	CLK_CKDIVR Reset value	- 0	- 0	- 0	- 0	- 0	- 0	HSIDIV1 1	HSIDIV0 1
0x01 to 0x02	Reserved area (2 bytes)								
0x03	CLK_PCKENR1 Reset value	PCKENR17 0	PCKENR16 0	PCKENR15 0	PCKENR14 0	PCKENR13 0	PCKENR12 0	PCKENR11 0	PCKENR10 0
0x04	CLK_PCKENR2 Reset value	- 0							PCKENR20 1
0x05	CLK_CCOR Reset value	- 0	- 0	CCOSLP1 0	CCOSLP0 1	CCOSEL2 0	CCOSEL1 0	CCOSEL0 0	CCOEN 0

1. Please refer to the “general hardware register map” table in the datasheet for details on register addresses.

9 Power management

By default, after a system or power reset, the microcontroller is in Run mode. In this mode the CPU is clocked by f_{MASTER} and executes the program code, while the system clock to all peripherals is gated off by the peripheral clock gating system.

While in Run mode, still keeping the CPU running and executing code, the application has several ways to reduce power consumption, such as:

- Slowing down the system clock
- Gating the clocks to individual peripherals when they are unused
- Switching off any unused analog functions

However, when the CPU does not need to be kept running, three dedicated low power modes can be used:

- Wait (wait for interrupt and wait for event)
- Active-halt
- Halt

One of these three modes can be selected and configured to obtain the best compromise between lowest power consumption, fastest startup time and available wakeup sources.

9.1 General considerations

Low power consumption features are generally very important for all types of application for energy saving. Ultra low power features are especially important for mobile applications to ensure long battery lifetimes. This is also crucial for environmental protection.

In a silicon chip there are two kind of consumption:

- **Static power consumption** which is due to analog polarization and leakage. Consumption due to leakage is so small that it is only significant in Halt and Active-halt modes (*Section 9.4: Low power modes*).
- **Dynamic power consumption** which comes from running the digital parts of the chip. It depends on V_{DD}, clock frequency and load capacitors.

In a microcontroller device the consumption depends on:

- V_{DD} supply voltage
- MCU size or number of digital gates (leakages and load capacitors)
- Clock frequency
- Number of active peripherals
- Available low power modes and low power levels

Device processing performance is also very important, as this allows the application to minimize the time spent in Run mode and maximize the time in Low power mode.

Using the MCU's flexible power management features, you can obtain a range of significant power savings while the system is running or able to resume operations quickly.

9.2 Managing the clock for low consumption

9.2.1 Slowing the system clocks

In run mode, choosing the clock frequency is very important to ensure the best compromise between performance and consumption. The selection is done by programming the CLK_CKDIVR register. Refer to *8.4.1: Clock divider register (CLK_CKDIVR) on page 63*.

Note: In applications where the MCU can be halted for certain periods, the power consumption can be minimized by keeping a fast clock (high performance execution) during active periods, in order to reduce the ratio between active periods and Halt (that is "zero"-consumption) periods.

9.2.2 Peripheral clock gating

For additional power saving you can use Peripheral Clock Gating (PCG). This can be done at any time by selectively enabling or disabling the f_{MASTER} clock connection to individual peripherals. Refer to *8.4.2: Peripheral clock gating register 1 (CLK_PCKENR1) on page 64* and *8.4.3: Peripheral clock gating register 2 (CLK_PCKENR2) on page 64*.

These settings are effective in both Run and Wait modes. Refer to *8.1.1: Peripheral clock gating (PCG) on page 62*.

Each PCG state represents a specific power or low power level.

9.3 Switching peripherals off

Any of the MCU peripherals can be deactivated when they are not used in order to minimize power consumption. This is true for both analog and digital peripherals.

Each ON/OFF combination represents a specific power or low power level.

Each of the peripherals can be switched off by a dedicated control bit:

- Single Wire Interface Module (SWIM) by the SWD bit in the SWIM_CSR register
- Auto-Wakeup Unit (AWU) by the AWUEN bit in the AWU_CSR register
- Beeper by the BEEPEN bit in the AWU_CSR register
- Timers by the CEN bits in the TIMx_CR1 registers
- I2C by the PE bit in the I2C_CR1 register
- SPI by the SPE bit in the SPI_CR1
- USART by the USART_CR2 register
- WWDG by the WWDG_CR register
- ProxSense by the PXS_CR1 register

Note: These control bits should be written only when the corresponding peripheral clock is enabled
Once the IWDG is activated, it cannot be disabled except with a reset.

9.4 Low power modes

By default, the microcontroller is in run mode after a system or power reset. However the device supports three low power modes to achieve the best compromise between low power consumption, short startup time and available wakeup sources:

- **Wait mode**: The CPU clock is stopped, but selected peripherals keep running. An internal or external interrupt or a Reset can be used to exit the microcontroller from Wait mode. Refer to *Section 9.4.1: Wait mode on page 70*
- **Active-halt mode**: The CPU and peripheral clocks are stopped, except LSI. The wakeup can be triggered by AWU or PXS interrupts, external interrupts or reset.
- **Halt mode**: The CPU and peripheral clocks are stopped, the device remains powered on. The wakeup is triggered by an external interrupt or reset. A few peripherals also have wakeup from Halt capability.

The main characteristics of the three low power modes are summarized in *Table 17*.

Table 17. Low power mode management

Mode (consumption level)		Main voltage regulator (MVR)	Oscillators	CPU	Peripherals	Wakeup trigger event
Wait (-)	WFI	ON	ON	OFF	ON(1)	All internal or external interrupts, reset
	WFE	ON	ON	OFF	ON(1)	All internal or external interrupts(2), WFE events, reset
Active-halt (- -)	AWU	OFF (LPVR ON)	HSI OFF, LSI ON	OFF	Only AWU, BEEP and IWDG if activated and if “no watchdog in Halt” option disabled	AWU or external(3) interrupts, reset
	PXS	ON	HSI OFF, LSI OFF, HSI_PXS ON	OFF	PXS	PXS, external(3) interrupts or reset
Halt (- - -)		OFF (LPVR ON)	LSI ON if IWDG activated and if "no watchdog in Halt" option is disabled.	OFF	Only BEEP and IWDG if activated and if “no watchdog in Halt” option disabled	External interrupts(3), reset

1. If the peripheral clock is not disabled by Peripheral Clock Gating function.
2. Refer to*: Wait for event (WFE) mode on page 70*
3. Including communication peripheral interrupts (see interrupt vector table).

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

- Slowing down the system clocks
- Gating the peripherals clocks when they are unused

Refer to *Section 9.2: Managing the clock for low consumption on page 68*.

9.4.1 Wait mode

Wait mode is entered from run mode by executing a WFI (Wait For Interrupt) or WFE (Wait For Event) instruction: this stops the CPU but allows the other peripherals and interrupt controller to continue running. The consumption decreases accordingly. Wait mode can be combined with PCG to further reduce power consumption of the device. Refer to *Section 8.1.1: Peripheral clock gating (PCG) on page 62*.

In Wait mode, all the registers and RAM contents are preserved and the clock configuration selected through the *Clock divider register (CLK_CKDIVR)* remains unchanged.

Wait for interrupt (WFI) mode

When an internal or external interrupt request occurs, the CPU wakes up from WFI mode, services the interrupt and resumes processing.

Note: In an interrupt based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the CFG_GCR register. Setting this bit causes the CPU to return to WFI mode without restoring the main execution context. This saves power by removing both the save/restore context activity and the need for a main software loop execution for power management (in order to return to WFI mode).

Wait for event (WFE) mode

There are two ways to wake up the CPU from WFE mode (see *Table 17.: Low power mode management on page 69*):

- When an interrupt occurs:
  - the CPU wakes up from WFE mode and services the interrupt. After processing the interrupt, the processor goes back to WFE mode.
- When a WFE event occurs:
  - the CPU wakes up and resumes processing. As the processing resumes directly after the WFE instruction, there is no context save/restore activity (this saves time and power consumption).

Note: In WFE mode, the interrupt sources are configured as external interrupts only if the corresponding status flags are cleared in the WFE_CR1 or WFE_CR2 register. Otherwise, they generate WFE events (no interrupt serviced and the user has to properly clear the corresponding status flags in the EXTI_SR1 or EXTI_SR2 register). Refer to Section 9.5: WFE registers on page 72, External interrupt status register 1 (EXTI_SR1) on page 53 and External interrupt status register 2 (EXTI_SR2) on page 54.

Further power consumption reduction may be achieved using this mode together with execution from RAM. In some very low power applications, when the main software routine is short and has a low execution time, this routine can be moved to RAM and executed from RAM. As the Flash program memory is not used at wakeup, the power consumption is then reduced during run time.

At any moment, another routine (stored in the Flash program memory) can be executed by software by simply calling/jumping to this routine.

9.4.2 Halt mode

In this mode the master clock is stopped. This means that the CPU and all the peripherals clocked by f_{MASTER} or by derived clocks are disabled. As a result, none of the peripherals are clocked and the digital part of the MCU consumes almost no power.

The Main Voltage Regulator (MVR) is switched off automatically when the MCU enters Halt mode. The MCU core is powered only by the Low Power Voltage Regulator (LPVR) and all the registers and RAM contents are preserved.

The MCU enters Halt mode when a HALT instruction is executed. Wakeup from Halt mode is triggered by an external interrupt, sourced by a general purpose I/O port configured as interrupt input or by an alternate function pin capable of triggering a peripheral interrupt.

In an interrupt based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the CPU configuration register. Setting this bit causes the CPU to return to Halt mode when executing the return from interrupt, without restoring the main execution context.

This saves power by removing the save/restore context activity and by removing the need to execute a main level software loop for power management (in order to return to WFI mode).

Caution: Before entering Halt mode, the ProxSense peripheral must be disabled or in Low power mode (PXS_EN = ‘0’ or LOW_POWER = ‘1’ in register PXS_CR1).

9.4.3 Active-halt mode

Active-halt mode is similar to Halt mode except that it does not require an external interrupt for wake up. It uses the AWU to generate a wakeup event internally after a programmable delay or the PXS to generate a wakeup event after a PXS interrupt occurrence.

Active-halt mode with AWU

To enter Active-halt mode with AWU, first enable the AWU as described in the AWU section. Then execute a HALT instruction.

In Active-halt mode with AWU, the main oscillator, the CPU and almost all peripherals are stopped. Only the LSI RC oscillator is running to drive the AWU counters, BEEP and IWDG counter if enabled.

The Main Voltage Regulator (MVR) is switched off automatically when the MCU enters Active-halt mode. The MCU core is powered only by the Low Power Voltage Regulator (LPVR) and all registers and RAM contents are preserved.

In order to reduce the consumption when using the Active-halt mode with AWU, the ProxSense must be disabled (PXS_EN = ‘0’ in register PXS_CR1) or in Low power mode (LOW_POWER = ‘1’ in register PXS_CR1).

Active-halt mode with ProxSense

To enter Active-halt mode with ProxSense, an acquisition must be in progress (CIPF = ‘1’ in register PXS_ISR). If an acquisition is pending on a synchronization signal, the device does not enter Active-halt mode with ProxSense if the synchronization trigger does not occur before the Halt instruction execution. This can be checked by polling the CIPF bit in the PXS_ISR register.

In Active-halt mode with ProxSense, the HSI and LSI oscillators, the CPU and almost all the peripherals are stopped. Only the HSI_PXS oscillator is running to drive the analog ProxSense system.

The Main Voltage Regulator (MVR) is always kept ON, the MCU core clock is switched OFF and all registers and RAM contents are preserved.

9.5 WFE registers

These registers are used to configure different interrupt sources as external interrupts or WFE events. Refer to *Section : Wait for event (WFE) mode on page 70*

9.5.1 WFE control register 1 (WFE_CR1)

Address Offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
EXTI_EV3	EXTI_EV2	EXTI_EV1	EXTI_EV0	PXS_EV	Reserved	TIM2_EV1	TIM2_EV0
rw	rw	rw	rw	rw		rw	rw

Bit 7 **EXTI_EV3**: External interrupt event 3
0: Interrupt sources from pin 3 of all ports generate external interrupts.
1: Interrupt sources from pin 3 of all ports generate WFE events (no interrupt serviced).
This bit is written by software.

Bit 6 **EXTI_EV2**: External interrupt event 2
0: Interrupt sources from pin 2 of all ports generate external interrupts.
1: Interrupt sources from pin 2 of all ports generate WFE events (no interrupt serviced).
This bit is written by software.

Bit 5 **EXTI_EV1**: External interrupt event 1
0: Interrupt sources from pin 1 of all ports generate external interrupts.
1: Interrupt sources from pin 1 of all ports generate WFE events (no interrupt serviced).
This bit is written by software.

Bit 4 **EXTI_EV0**: External interrupt event 0
0: Interrupt sources from pin 0 of all ports generate external interrupts.
1: Interrupt sources from pin 0 of all ports generate WFE events (no interrupt serviced).
This bit is written by software.

Bit 3 **PXS_EV**: ProxSense interrupt
0: PXS interrupt sources generate external interrupts.
1: PXS interrupt sources logically ORed and configured to generate WFE events (no interrupt serviced).
This bit is written by software.

Bit 2 Reserved

Bit 1 **TIM2_EV1**: TIM2 event 1

0: TIM2 capture and compare interrupt sources generate external interrupts.

1: TIM2 capture and compare interrupt sources logically ORed and configured to generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 0 **TIM2_EV0**: TIM2 event 0

0: TIM2 update, trigger and break interrupt sources generate external interrupts.

1: TIM2 update, trigger and break interrupt sources logically ORed and configured to generate WFE events (no interrupt serviced).

This bit is written by software.

9.5.2 WFE control register 2 (WFE_CR2)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		EXTI_EVD	EXTI_EVB	EXTI_EV7	EXTI_EV6	EXTI_EV5	EXTI_EV4
		rw	rw	rw	rw	rw	rw

Bits 7:6 Reserved

Bit 5 **EXTI_EVD**: External interrupt event on Port D

0: Interrupt sources from Port D generate external interrupts.

1: Interrupt sources from Port D generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 4 **EXTI_EVB**: External interrupt event on Port B

0: Interrupt sources from Port B generate external interrupts.

1: Interrupt sources from Port B generate WFE events (no interrupt serviced).

This bit is written by software

Bit 3 **EXTI_EV7**: External interrupt event 7

0: Interrupt sources from pin 7 of all ports generate external interrupts.

1: Interrupt sources from pin 7 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 2 **EXTI_EV6**: External interrupt event 6

0: Interrupt sources from pin 6 of all ports generate external interrupts.

1: Interrupt sources from pin 6 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 1 **EXTI_EV5**: External interrupt event 5

0: Interrupt sources from pin 5 of all ports generate external interrupts.

1: Interrupt sources from pin 5 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

Bit 0 **EXTI_EV4**: External interrupt event 4

0: Interrupt sources from pin 4 of all ports generate external interrupts.

1: Interrupt sources from pin 4 of all ports generate WFE events (no interrupt serviced).

This bit is written by software.

9.6 WFE register map and reset values

Table 18. WFE register map

Address offset (1)	Register name	7	6	5	4	3	2	1	0
0x00	WFE_CR1 Reset value	EXTI_EV3 0	EXTI_EV2 0	EXTI_EV1 0	EXTI_EV0 0	PXS_EV 0	- 0	TIM2_EV1 1	TIM2_EV0 1
0x01	WFE_CR2 Reset value	- 00		EXTI_EVD 0	EXTI_EVB 0	EXTI_EV7 0	EXTI_EV6 0	EXTI_EV5 0	EXTI_EV4 0

1. Please refer to the “general hardware register map” table in the datasheet for details on register addresses.

10 General purpose I/O ports (GPIO)

10.1 Introduction

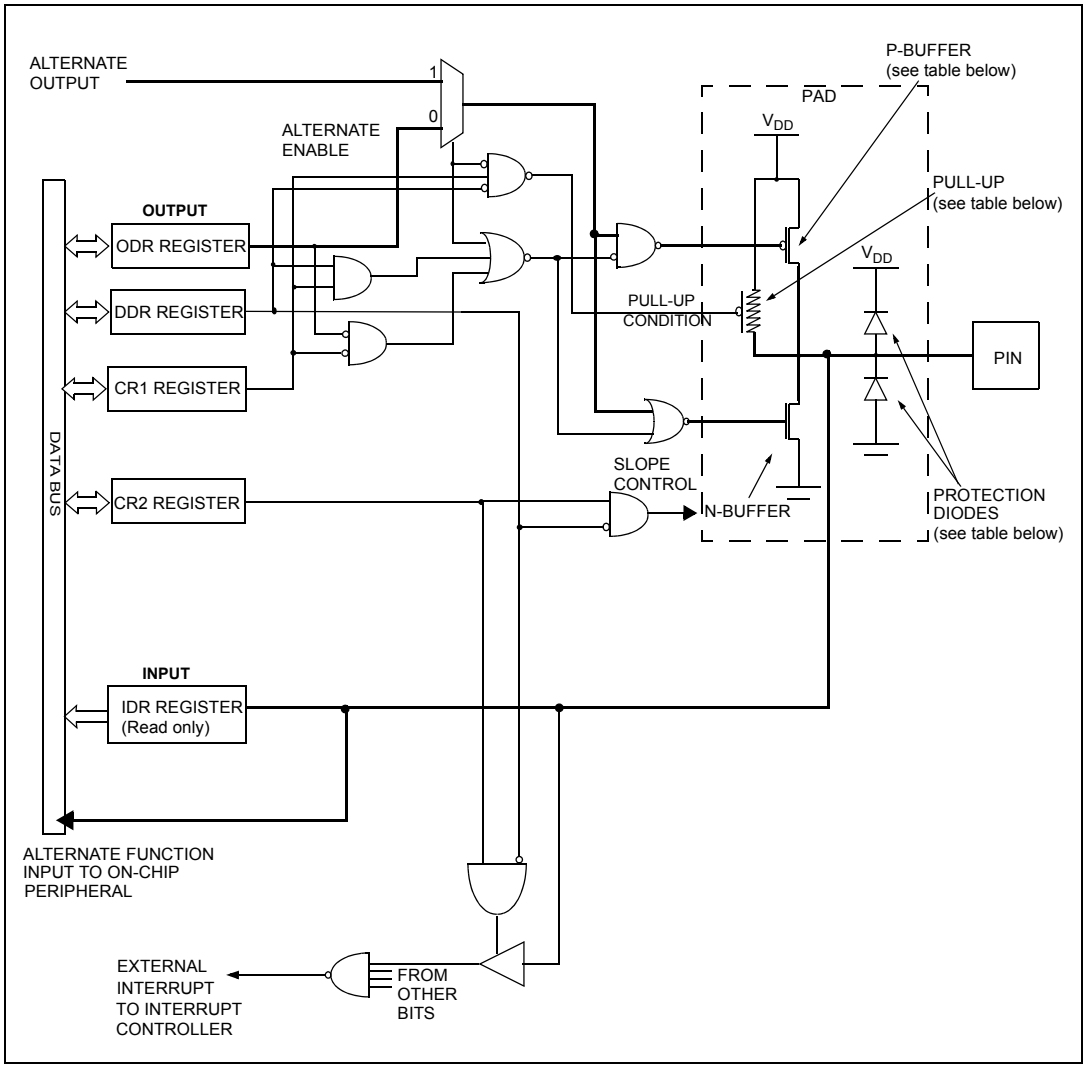
General purpose input/output ports are used for data transfers between the chip and the external world. An I/O port can contain up to eight pins. Each pin can be individually programmed as a digital input or digital output. In addition, some ports may have alternate functions like analog inputs, external interrupts, input/output for on-chip peripherals. Only one alternate function can be mapped to a pin at a time.

An output data register, input data register, data direction register and two configuration registers are associated with each port. A particular port will behave as an input or output depending on the status of the data direction register of the port.

10.2 GPIO main features

- Port bits can be configured individually
- Selectable input modes: floating input or input with pull-up
- Selectable output modes: push-pull output or pseudo-open-drain.
- Separate registers for data input and output
- External interrupts can be enabled and disabled individually
- Output slope control for reduced EMC noise
- Alternate function I/Os for on-chip peripherals
- Read-modify-write possible on data output latch
- I/O state guaranteed in voltage range 1.6 V to $V_{DDIOmax}$

Figure 13. GPIO block diagram



10.3 Port configuration and usage

An output data register (ODR), pin input register (IDR), data direction register (DDR) are always associated with each port.

The control register 1 (CR1) and control register 2 (CR2) allow input/output options. An I/O pin is programmed using the corresponding bits in the DDR, ODR, CR1 and CR2 registers.

Bit *n* in the registers corresponds to pin *n* of the Port.

The various configurations are summarized in *Table 19*.

Table 19. I/O port configuration summary

Mode	DDR bit	CR1 bit	CR2 bit	Function	Pull-up	P-buffer	Diodes	
							to V_{DD}	to V_{SS}
Input	0	0	0	Floating without interrupt	Off	Off	On	On
	0	1	0	Pull-up without interrupt	On			
	0	0	1	Floating with interrupt	Off			
	0	1	1	Pull-up with interrupt	On			
Output	1	0	0	Open drain output	Off	Off		
	1	1	0	Push-pull output		On		
	1	0	1	Open drain output, fast mode		Off		
	1	1	1	Push-pull, fast mode	Off	On		
	1	x	x	True open drain (on specific pins)	Not implemented		Not implemented (1)	

1. The diode connected to V_{DD} is not implemented in true open drain pads. A local protection between the pad and V_{OL} is implemented to protect the device against positive stress.

Warning: On some packages, some ports must be considered as active even if they do not exist on the package. To avoid spurious effects, configure them as pull-up inputs without interrupt at startup, and keep them in this state when changing the port configuration. Refer to the datasheet for additional information.

10.3.1 Input modes

Clearing the DDRx bit selects input mode. In this mode, reading a IDR bit returns the digital value of the corresponding I/O pin.

Refer to *Section 10.7: Input mode details on page 79* for information on analog input, external interrupts and Schmitt trigger enable/disable.

As shown in , four different input modes can be theoretically be configured by software: floating without interrupt, floating with interrupt, pull-up without interrupt or pull-up with interrupt. However in practice, not all ports have external interrupt capability or pull-ups. You should refer to the datasheet pin-out description for details on the actual hardware capability of each port.

10.3.2 Output modes

Setting the DDRx bit selects output mode. In this mode, writing to the ODR bits applies a digital value to the I/O through the latch. Reading IDR bit returns the digital value from the corresponding I/O pin. Using the CR1, CR2 registers, different output modes can be configured by software: Push-pull output, Open-drain output.

Refer to *Section 10.8: Output mode details on page 79* for more information.

10.4 Reset configuration

All I/O pins are generally input floating under reset (i.e. during the reset phase) and at reset state (i.e. after reset release). However, a few pins may have a different behavior. Refer to the datasheet pinout description for all details.

10.5 Unused I/O pins

Unused I/O pins must not be left floating to avoid extra current consumption. They must be put into one of the following configurations:

- connected to V_{DD} or V_{SS} by external pull-up or pull-down resistor and kept as input floating (reset state),
- configured as input with internal pull-up/down resistor,
- configured as output push-pull low.

The I/O ports not present on smaller packages are automatically configured by hardware (unless otherwise specified in the datasheet). As a consequence, no configuration is required on these I/O ports. The bits corresponding to these ports in the configuration registers Px_ODR, PxDDR, PxCR1 and PxCR2 can be written, but this will have no effect. The value read in the corresponding bits of the PxIDR register will be '0'.

10.6 Low power modes

Table 20. Effect of low power modes on GPIO ports

Mode	Description
Wait	No effect on I/O ports. External interrupts cause the device to exit from Wait mode.
Halt	No effect on I/O ports. External interrupts cause the device to wakeup from Halt mode.

10.7 Input mode details

10.7.1 Alternate function input

Some I/Os can be used as alternate function input. For example as the port may be used as the input capture input to a timer. Alternate function inputs are not selected automatically, you select them by writing to a control bit in the registers of the corresponding peripheral. For Alternate Function input, you should select floating or pull-up input configuration in the DDR and CR1 registers.

10.7.2 Interrupt capability

Each I/O can be configured as an input with interrupt capability by setting the CR2x bit while the I/O is in input mode. In this configuration, a signal edge or level input on the I/O generates an interrupt request.

Falling or rising edge sensitivity is programmed independently for each interrupt vector in the EXTI_CR[2:1] registers.

External interrupt capability is only available if the port is configured in input mode.

Interrupt masking

Interrupts can be enabled/disabled individually by programming the corresponding bit in the configuration register (Px_CR2). At reset state, the interrupts are disabled.

If a pin alternate function is TLI, use the Px_CR2 bit to enable/disable the TLI interrupt. The TLI interrupt is associated to a dedicated interrupt vector.

10.8 Output mode details

10.8.1 Alternate function output

Alternate function outputs provide a direct path from a peripheral to an output or to an I/O pad, taking precedence over the port bit in the data output latch register (Px_ODR) and forcing the Px_DDR corresponding bit to 1.

An alternate function output can be push-pull or pseudo-open drain depending on the peripheral and Control register 1 (Px_CR1) and slope can be controlled depending on the Control register 2 (Px_CR2) values.

Examples:

SPI outputs must be set-up as push-pull. The slope of SPI outputs is controlled by hardware and configured in fast mode to enable an optimal operation. The user must then keep the CR2 slope control bit cleared to avoid spurious interrupts.

10.8.2 Slope control

The maximum frequency that can be applied to an I/O can be controlled by software using the CR2 bit. Low frequency operation with improved EMC behavior is selected at reset. Higher frequency (up to 10 MHz) can be selected if needed. This feature can be applied in either open drain or push-pull output mode on I/O ports of output type O3 or O4. Refer to the pin description tables in the datasheets for the specific output type information for each pin.

10.9 GPIO registers

The bit of each port register drives the corresponding pin of the port.

10.9.1 Port x output data register (Px_ODR)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **ODR[7:0]**: Output data register bits

Writing to the ODR register when in output mode applies a digital value to the I/O through the latch. Reading the ODR returns the previously latched value in the register.

In Input mode, writing in the ODR register, latches the value in the register but does not change the pin state. The ODR register is always cleared after reset. Bit read-modify-write instructions (BSET, BRST) can be used on the DR register to drive an individual pin without affecting the others.

10.9.2 Port x pin input register (Px_IDR)

Address offset: 0x01
Reset value: 0xXX

7	6	5	4	3	2	1	0
IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r

Bits 7:0 **IDR[7:0]**: Pin input values

The pin register can be used to read the pin value irrespective of whether port is in input or output mode. This register is read-only.

0: Low logic level

1: High logic level

Note: Px_IDR reset value depends on the external circuitry.

10.9.3 Port x data direction register (Px_DDR)

Address offset: 0x02
Reset value: 0x00

7	6	5	4	3	2	1	0
DDR7	DDR6	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **DDR[7:0]**: Data direction bits

These bits are set and cleared by software to select input or output mode for a particular pin of a port.

0: Input mode

1: Output mode

10.9.4 Port x control register 1 (Px_CR1)

Address offset: 0x03
Reset value: 0x00 except for PA_CR1 which reset value is 0x01.

7	6	5	4	3	2	1	0
C17	C16	C15	C14	C13	C12	C11	C10
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **C1[7:0]**: Control bits

These bits are set and cleared by software. They select different functions in input mode and output mode (see .

– **In input mode (DDR = 0):**

0: Floating input

1: Input with pull-up

– **In output mode (DDR = 1):**

0: Pseudo open drain

1: Push-pull, slope control for the output depends on the corresponding CR2 bit

Note: This bit has no effect on true open drain ports (refer to pin marked “T” in datasheet pin description table).

10.9.5 Port x control register 2 (Px_CR2)

Address offset: 0x04
Reset value: 0x00

7	6	5	4	3	2	1	0
C27	C26	C25	C24	C23	C22	C21	C20
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **C2[7:0]**: Control bits

These bits are set and cleared by software. They select different functions in input mode and output mode. In input mode, the CR2 bit enables the interrupt capability if available. If the I/O does not have interrupt capability, setting the CR2 bit has no effect. In output mode, setting the bit increases the speed of the I/O. This applies to ports with O3 and O4 output types (see pin description table).

– **In input mode (DDR = 0):**
  0: External interrupt disabled
  1: External interrupt enabled
– **In output mode (DDR = 1):**
  0: Output speed up to 2 MHz
  1: Output speed up to 10 MHz

10.9.6 GPIO register map and reset values

Each GPIO port has five registers mapped as shown in *Table 21*. Refer to the register map in the corresponding datasheet for the base address for each port.

Note: At reset state, all ports are input floating. Exceptions are indicated in the pin description table of the corresponding datasheet.

Table 21. GPIO register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	Px_ODR Reset value	ODR7 0	ODR6 0	ODR5 0	ODR4 0	ODR3 0	ODR2 0	ODR1 0	ODR0 0
0x01	Px_IDR Reset value	IDR7 x	IDR6 x	IDR5 x	IDR4 x	IDR3 x	IDR2 x	IDR1 x	IDR0 x
0x02	Px_DDR Reset value	DDR7 0	DDR6 0	DDR5 0	DDR4 0	DDR3 0	DDR2 0	DDR1 0	DDR0 0
0x03	Px_CR1 Reset value	C17 0	C16 0	C15 0	C14 0	C13 0	C12 0	C11 0	C10 0
0x04	Px_CR2 Reset value	C27 0	C26 0	C25 0	C24 0	C23 0	C22 0	C21 0	C20 0

11 System configuration controller (SYSCFG)

11.1 Introduction

The system configuration controller offers remapping capabilities of TIM3 channels on different I/O ports. To use an alternate function, the corresponding peripheral must be enabled in the peripheral registers. Alternate function remapping does not affect the GPIO capabilities of the I/O ports (see *Section 10: General purpose I/O ports (GPIO) on page 75*). Refer to *Section 11.2: SYSCFG register* for remapping capabilities on TIM3 channels.

11.2 SYSCFG register

11.2.1 SYSCFG remap control register 1 (SYSCFG_RMPCR1)

Address offset: 0x00 509E

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				TIM3_CH2	TIM3_CH1	Reserved	
				rw	rw		

Bits 7:4 Reserved

Bit 3 **TIM3_CH2**:
0: Timer3 channel2 mapped on PD[7]
1: Timer3 channel2 mapped on PA[2]

Bit 2 **TIM3_CH1**:
0: Timer3 channel1 mapped on PD[6]
1: Timer3 channel1 mapped on PA[1]

Bits 1:0 Reserved

11.2.2 SYSCFG register map and reset values

Table 22. Register map

Offset address	Register name	7	6	5	4	3	2	1	0
0x00	SYSCFG_RMPCR1	-	-	-	-	TIM3_CH2	TIM3_CH1	-	-
	Reset value	0	0	0	0	0	0	0	0

12 Auto-wakeup (AWU)

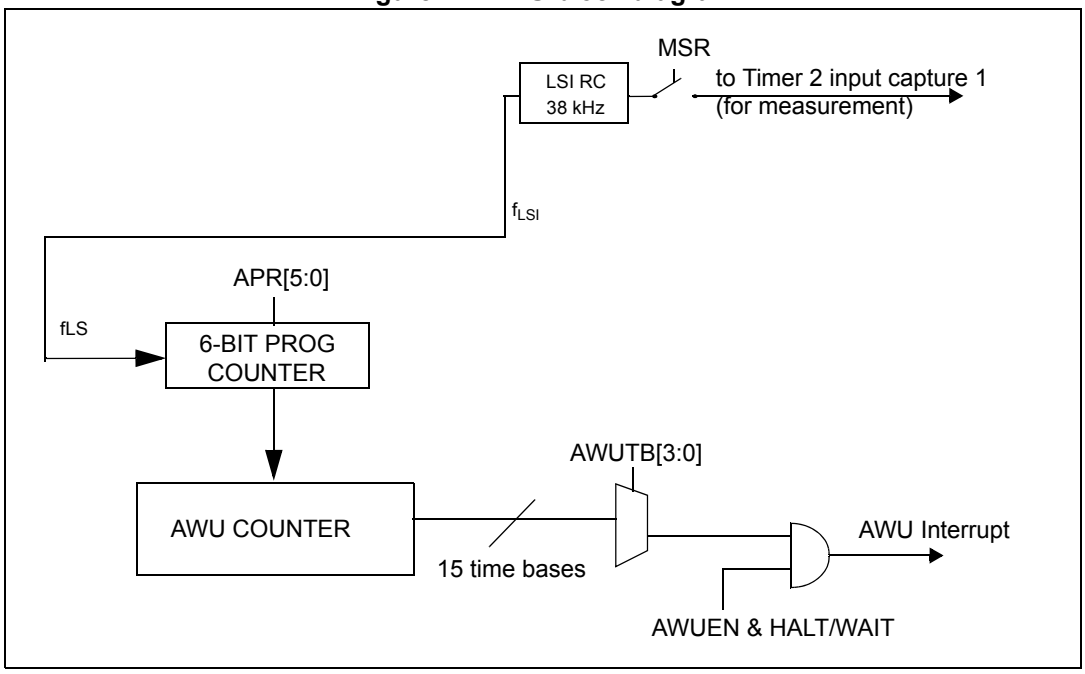
12.1 Introduction

The AWU is used to provide an internal wakeup time base that is used when the MCU goes into Active-halt power saving mode. This time base is clocked by the low speed internal (LSI) RC oscillator clock.

12.2 LSI clock measurement

To ensure the best possible accuracy when using the LSI clock, its frequency can be measured with TIM2 timer input capture 1.

Figure 14. AWU block diagram



12.3 AWU functional description

12.3.1 AWU operation

To use the AWU, perform the following steps in order:

1. Measure the LS clock frequency using the MSR bit in AWU_CSR register and TIM input capture 1.
2. Define the appropriate prescaler value by writing to the APR [5:0] bits in the *Asynchronous prescaler register (AWU_APR)*.
3. Select the desired auto-wakeup delay by writing to the AWUTB[3:0] bits in the *Timebase selection register (AWU_TBR)*.
4. Set the AWUEN bit in the *Control/status register (AWU_CSR)*.
5. Execute the HALT instruction. AWU counters are reloaded and start to count a new AWU time interval.

Note: The counters only start when the MCU enters Active-halt mode after a HALT instruction (refer to the Active-halt mode section in the power management chapter). The AWU interrupt is then enabled at the same time.

The prescaler counter starts to count only if APR[5:0] value is different from its reset value, 0x3F.

Idle mode

If the AWU is not in use, then the AWUTB[3:0] bits the *Timebase selection register (AWU_TBR)* should be loaded with 0b0000 to reduce power consumption.

12.3.2 Time base selection

Please refer to the *Asynchronous prescaler register (AWU_APR)* and *Timebase selection register (AWU_TBR)* descriptions.

The AWU time intervals depend on the values of:

- AWUTB[3:0] bits. This gives the counter output rank.
- APR[5:0] bits. This gives the prescaler division factor (APR_{DIV}).

15 non-overlapped ranges of time intervals can be defined as follows:

Table 23. Time base calculation table

Interval range		AWUTB[3:0]	APR_{DIV} formula for time interval calculation	APR_{DIV} range
f_{LS} = f	f_{LS} = 38 kHz			
2/f - 64/f	0.53 ms - 1.68 ms	0001	APR_{DIV}/f_{LS}	2 to 64
2x32/f - 2x2x32/f	1.68 ms - 3.37 ms	0010	$2 \times APR_{DIV}/f_{LS}$	32 to 64
2x64/f - 2x2x64/f	3.37 ms - 6.74 ms	0011	$2^2 \times APR_{DIV}/f_{LS}$	32 to 64
2^2x64/f - 2^2x128/f	6.74 ms - 13.47 ms	0100	$2^3 \times APR_{DIV}/f_{LS}$	32 to 64
2^3x64/f - 2^3x128/f	13.47 ms - 26.95 ms	0101	$2^4 \times APR_{DIV}/f_{LS}$	32 to 64
2^4x64/f - 2^4x128/f	26.95 ms - 53.89 ms	0110	$2^5 \times APR_{DIV}/f_{LS}$	32 to 64
2^5x64/f - 2^5x128/f	53.89 ms - 107.8 ms	0111	$2^6 \times APR_{DIV}/f_{LS}$	32 to 64
2^6x64/f - 2^6x128/f	107.8 ms - 215.6 ms	1000	$2^7 \times APR_{DIV}/f_{LS}$	32 to 64
2^7x64/f - 2^7x128/f	215.6 ms - 431.2 ms	1001	$2^8 \times APR_{DIV}/f_{LS}$	32 to 64
2^8x64/f - 2^8x128/f	431.2 ms - 862.3 ms	1010	$2^9 \times APR_{DIV}/f_{LS}$	32 to 64
2^9x64/f - 2^9x128/f	862.3 ms - 1.72 s	1011	$2^{10} \times APR_{DIV}/f_{LS}$	32 to 64
2^{10}x64/f - 2^{10}x128/f	1.72 s - 3.45 s	1100	$2^{11} \times APR_{DIV}/f_{LS}$	32 to 64
2^{11}x64/f - 2^{11}x128/f	3.45 s - 6.90 s	1101	$2^{12} \times APR_{DIV}/f_{LS}$	32 to 64
2^{11}x130/f - 2^{11}x320/f	7.01 s - 17.25 s	1110	$5 \times 2^{11} \times APR_{DIV}/f_{LS}$	26 to 64
2^{11}x330/f - 2^{12}x960/f	17.79 s - 103.48 s	1111	$30 \times 2^{11} \times APR_{DIV}/f_{LS}$	11 to 64

In order to obtain the right values for AWUTB[3:0] and APR_{DIV}, you have to:

- First, search the interval range corresponding to the desired time interval. This gives the AWUTB[3:0] value.
- Then APR_{DIV} can be chosen to obtain a time interval value as close as possible to the desired one. This can be done using the formulas listed in the table above.

Note: If the target value is between $2^{11}x128/f_{LS}$ and $2^{11}x130/f_{LS}$ or between $2^{11}x320/f_{LS}$ and $2^{11}x330/f_{LS}$, the value closer to the target one must be chosen.

Example 1

- f_{LS} = 38 kHz
- Target time interval = 16 ms

The appropriate interval range is: 13.47 ms - 26.95 ms
so the AWUTB[3:0] value is 0x5.

The APR_{DIV} value is:
16 ms = 2^4 x APR_{DIV} / f_{LS} => APR_{DIV} = ($16*10^{-3}$ x f_{LS}) / 2^4 = 38
so the APR[5:0] value is 38 (0x26)

Example 2

- f_{LS} = 38 kHz
- Target time interval = 20 ms

The appropriate interval range is: 13.47 ms - 26.95 ms
So the AWUTB[3:0] value is 0x5.

The APR_{DIV} value is:
20 ms = 2^4 x APR_{DIV} / f_{LS} => APR_{DIV} = ($20*10^{-3}$ x f_{LS}) / 2^4 = 47.5
So the AWUTB[3:0] can be either 47 or 48 which gives a time base of 19.79 ms or 20.21 ms respectively. This is not exactly 20 ms.

12.3.3 LSI clock frequency measurement

Due to the oscillator frequency dispersion, to obtain a precise AWU time interval or beeper output, the exact LSI frequency has to be measured.

Use the following procedure:

1. Set the MSR bit in the *Control/status register (AWU_CSR)* to connect the LSI clock internally to a timer input capture.
2. Measure the frequency of the LSI clock using the Timer input capture interrupt.
3. Write the appropriate value in the APR [5:0] bits in the *Asynchronous prescaler register (AWU_APR)* to adjust the AWU time interval to the desired length. The AWUTB[3:0] bits can be modified to select different time intervals.

LSI clock frequency measurement can also be used to calibrate the beeper frequency (see *Section 13.2.2*).

12.4 AWU registers

12.4.1 Control/status register (AWU_CSR)

Address offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		AWUF	AWUEN	Reserved			MSR
		rc_r	rw				rw

Bits 7:6 Reserved

Bit 5 **AWUF**: Auto-wakeup flag

This bit is set by hardware when the AWU module generates an interrupt and cleared by reading the AWU_CSR register. Writing to this bit does not change its value.

0: No AWU interrupt occurred
1: AWU interrupt occurred

Bit 4 **AWUEN**: Auto-wakeup enable

This bit is set and cleared by software. It enables the auto-wakeup feature. If the microcontroller enters Active-halt or Wait mode, the AWU feature wakes up the microcontroller after a programmable time delay.

0: AWU (Auto-wakeup) disabled
1: AWU (Auto-wakeup) enabled

Bits 3:1 Reserved

Bit 0 **MSR**: Measurement enable

This bit connects the f_{LS} clock to a timer input capture. This allows the timer to be used to measure the LS frequency (f_{LS}).

0: Measurement disabled
1: Measurement enabled

Note: Refer to the datasheet for information on which timer input capture can be connected to the LSI clock in the specific product).

12.4.2 Asynchronous prescaler register (AWU_APR)

Address offset: 0x01

Reset value: 0x3F

7	6	5	4	3	2	1	0
Reserved		APR[5:0]					
		rw					

Bits 7:6 Reserved

Bits 5:0 **APR[5:0]**: Asynchronous prescaler divider

These bits are written by software to select the prescaler divider (APR_{DIV}) feeding the counter clock.

0x00: APR_{DIV} = 2 0x0E: APR_{DIV} = 16
0x01: APR_{DIV} = 3 0x0F: APR_{DIV} = 17
... ...
0x06: APR_{DIV} = 8 0x3E: APR_{DIV} = 64

Note: This register must not be kept at its reset value (0x3F)

12.4.3 Timebase selection register (AWU_TBR)

Address offset: 0x02

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				AWUTB[3:0]			
				rw			

Bits 7:4 Reserved

Bits 3:0 **AWUTB[3:0]**: Auto-wakeup timebase selection

These bits are written by software to define the time interval between AWU interrupts. AWU interrupts are enabled when AWUEN = 1.

0000: No interrupt

0001: APR_{DIV}/f_{LS} 0010: $2xAPR_{DIV}/f_{LS}$ 0011: $2^2APR_{DIV}/f_{LS}$

0100: $2^3APR_{DIV}/f_{LS}$ 0101: $2^4APR_{DIV}/f_{LS}$ 0110: $2^5APR_{DIV}/f_{LS}$

0111: $2^6APR_{DIV}/f_{LS}$ 1000: $2^7APR_{DIV}/f_{LS}$ 1001: $2^8APR_{DIV}/f_{LS}$

1010: $2^9APR_{DIV}/f_{LS}$ 1011: $2^{10}APR_{DIV}/f_{LS}$ 1100: $2^{11}APR_{DIV}/f_{LS}$

1101: $2^{12}APR_{DIV}/f_{LS}$ 1110: $5x2^{11}APR_{DIV}/f_{LS}$ 1111: $30x2^{11}APR_{DIV}/f_{LS}$

12.4.4 AWU register map and reset values

Table 24. AWU register map

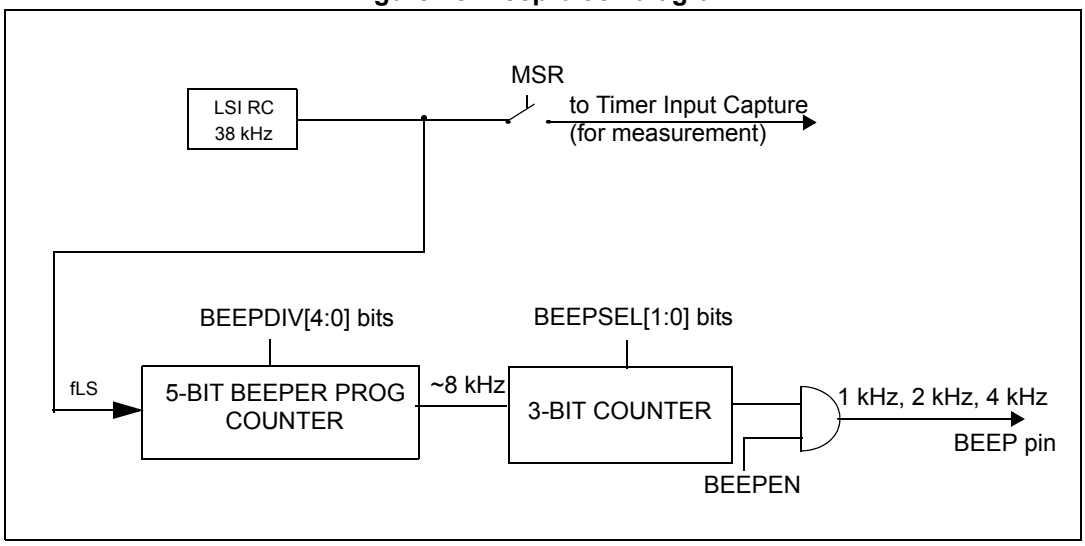
Address offset	Register name	7	6	5	4	3	2	1	0
0x00	AWU_CSR Reset value	- 0	- 0	AWUF 0	AWUEN 0	- 0	- 0	- 0	MSR 0
0x01	AWU_APR Reset value	- 0	- 0	APR5 1	APR4 1	APR3 1	APR2 1	APR1 1	APR0 1
0x02	AWU_TBR Reset value	- 0	- 0	- 0	- 0	AWUTB3 0	AWUTB2 0	AWUTB1 0	AWUTB0 0

13 Beeper (BEEP)

13.1 Introduction

This function generates a beep signal in the range of 1, 2 or 4 kHz when the LS clock is operating at a frequency of 38 kHz.

Figure 15. Beep block diagram



13.2 Beeper functional description

13.2.1 Beeper operation

To use the beep function, perform the following steps in order:

1. Calibrate the LS clock frequency as described in *Section 13.2.2: Beeper calibration* to define BEEPDIV[4:0] value.
2. Select 1 kHz, 2 kHz or 4 kHz output frequency by writing to the BEEPSEL[1:0] bits in the *Beeper control/status register (BEEP_CSR)*.
3. Set the BEEPEN bit in the *Beeper control/status register (BEEP_CSR)* to enable the LS clock source.

Note: The prescaler counter starts to count only if BEEPDIV[4:0] value is different from its reset value, 0x1F.

13.2.2 Beeper calibration

This procedure can be used to calibrate the LS kHz clock in order to reach the standard frequency output, 1 kHz, 2 kHz or 4 kHz.

Use the following procedure:

1. Measure the LSI clock frequency (refer to *Section 12.3.3: LSI clock frequency measurement* above).
2. Calculate the $BEEP_{DIV}$ value as follows, where A and x are the integer and fractional part of $f_{LS}/8$ (in kHz):

   $BEEP_{DIV}$ = A-2 when x is less than or equal to A/(1+2*A), else

   $BEEP_{DIV}$ = A-1
3. Write the resulting $BEEP_{DIV}$ value in the BEEPDIV[4:0] bits in the *Beeper control/status register (BEEP_CSR)*.

13.3 Beeper registers

13.3.1 Beeper control/status register (BEEP_CSR)

Address offset: 0x00
Reset value: 0x1F

7	6	5	4	3	2	1	0
BEEPSEL[1:0]		BEEPEN	BEEPDIV[4:0]				
rw		rw	rw				

Bits 7:6 **BEEPSEL[1:0]**: Beep selection

These bits are set and cleared by software to select 1, 2 or 4 kHz beep output when calibration is done.

00: $f_{LS}/(8 \times BEEP_{DIV})$ kHz output
01: $f_{LS}/(4 \times BEEP_{DIV})$ kHz output
1x: $f_{LS}/(2 \times BEEP_{DIV})$ kHz output

Bit 5 **BEEPEN**: Beep enable

This bit is set and cleared by software to enable the beep feature.

0: Beep disabled
1: Beep enabled

Bits 4:0 **BEEPDIV[4:0]**: Beep prescaler divider

These bits are set and cleared by software to define the Beeper prescaler dividing factor $BEEP_{DIV}$.

0x00: $BEEP_{DIV}$ = 2
0x01: $BEEP_{DIV}$ = 3
...
0x0E: $BEEP_{DIV}$ = 16
0x0F: $BEEP_{DIV}$ = 17
0x1E: $BEEP_{DIV}$ = 32

Note: This register must not be kept at its reset value (0x1F)

13.3.2 Beeper register map and reset values

Table 25. Beeper register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	BEEP_CSR Reset value	BEEPSEL[2:0] 00		BEEPEN 0	BEEPDIV[4:0] 11111				

14 Window watchdog (WWDG)

14.1 Introduction

The window watchdog is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence. The watchdog circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the contents of the downcounter before the T6 bit becomes cleared. An MCU reset is also generated if the 7-bit downcounter value (in the control register) is refreshed before the downcounter has reached the window register value. This implies that the counter must be refreshed in a limited window.

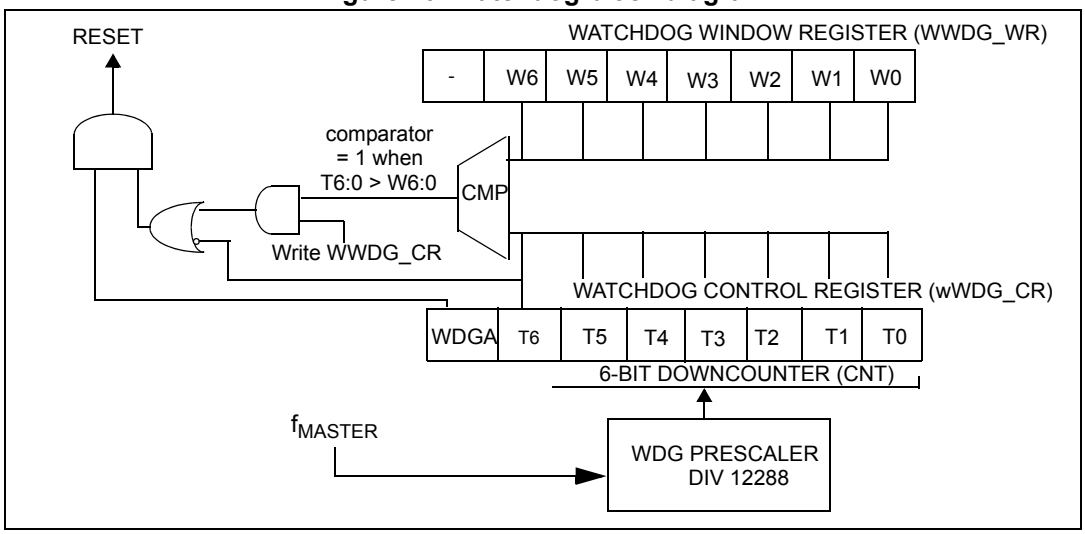
14.2 WWDG main features

- Programmable free-running downcounter
- Conditional reset
  - Reset (if watchdog activated) when the downcounter value becomes less than 0x40
  - Reset (if watchdog activated) if the downcounter is reloaded outside the window (see *Figure 17*)
- Hardware/software watchdog activation (selectable by option byte)
- Optional reset on HALT instruction (configurable by option byte)

14.3 WWDG functional description

If the watchdog is activated (the WDGA bit is set) and when the 7-bit downcounter (T[6:0] bits) rolls over from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset cycle pulling low the reset pin. If the software refreshes the counter while the counter is greater than the value stored in the window register, then a reset is generated.

Figure 16. Watchdog block diagram



The application program must write in the WWDG_CR register at regular intervals during normal operation to prevent an MCU reset. This operation must occur only when the counter value is lower than the window register value. The value to be stored in the WWDG_CR register must be between 0xFF and 0xC0:

- Enabling the watchdog:

  When software watchdog is selected (by option byte), the watchdog is 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.

  When hardware watchdog is selected (by option byte), the watchdog is always active and the WDGA bit is not used.
- Controlling the downcounter:

  This downcounter is free-running: It counts down even if the watchdog is disabled. When the watchdog is enabled, the T6 bit must be set to prevent generating an immediate reset.

  The T[5:0] bits contain the number of increments which represents the time delay before the watchdog produces a reset. The timing varies between a minimum and a maximum value due to the unknown status of the prescaler when writing to the WWDG_CR register (see *Figure 17*).

  The window register (WWDG_WR) contains the high limit of the window: To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x3F. *Figure 17* 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).

- Watchdog reset on halt option
  If the watchdog is activated and the watchdog reset on halt option is selected, then the HALT instruction will generate a reset.

14.4 How to program the watchdog timeout

The formula below can be used to calculate the WWDG timeout, t_{WWDG}, expressed in ms:

$$t_{WWDG} = T_{f_{MASTER}} \times 12288 \times (T[5:0] + 1)$$

where $T_{fMASTER}$ is the peripheral clock period expressed in ms

Warning: When writing to the WWDG_CR register, always write 1 in the T6 bit to avoid generating an immediate reset.

Figure 17. Window watchdog timing diagram

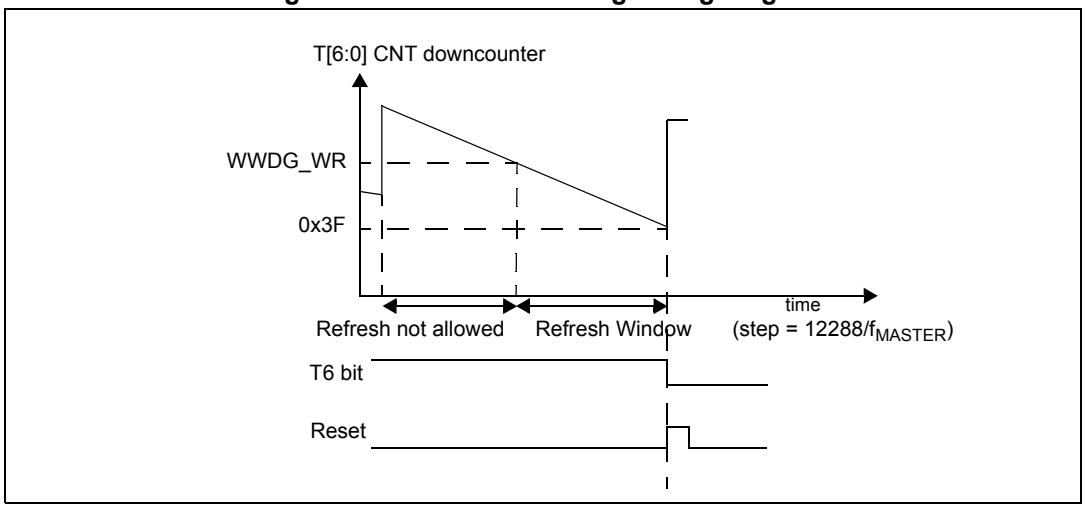


Table 26. Window watchdog timing example

T[6:0]	f_{MASTER} (MHz)	
	2	16
40h	6.144	0.768
7Fh	393.216	49.152

14.5 WWDG low power modes

Table 27. Effect of low power modes on WWDG

Mode	Description	
Wait	No effect on watchdog: The downcounter continues to decrement.	
Halt	WWDG_HALT in option byte	
	0	No watchdog reset is generated. The MCU enters Halt mode. The watchdog counter is decremented once and then stops counting and is no longer able to generate a watchdog reset until the MCU receives an external interrupt or a reset. If an interrupt is received (refer to interrupt table mapping to see interrupts which can occur in Halt mode), the watchdog restarts counting after the stabilization delay. If a reset is generated, the watchdog is disabled (reset state) unless hardware watchdog is selected by option byte. For application recommendations see *Section 14.7* below.
	1	A reset is generated instead of entering Halt mode.
Active-halt	x	No reset is generated. The MCU enters Active-halt mode. The watchdog counter is not decremented. It stops counting. When the MCU receives an oscillator interrupt or external interrupt, the watchdog restarts counting immediately. When the MCU receives a reset the watchdog restarts counting after the stabilization delay.

14.6 Hardware watchdog option

If hardware watchdog is selected by option byte, the watchdog is always active and the WDGA bit in the WWDG_CR register is not used. Refer to the option byte description in the datasheet.

14.7 WWDG interrupts

None.

14.8 WWDG registers

14.8.1 Control register (WWDG_CR)

Address offset: 0x00
Reset value: 0x7F

7	6	5	4	3	2	1	0
WDGA	T6	T5	T4	T3	T2	T1	T0
rs	rw	rw	rw	rw	rw	rw	rw

Bit 7 **WDGA**: Activation bit[1]
This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.
0: Watchdog disabled
1: Watchdog enabled

Bits 6:0 **T[6:0]**: 7-bit counter (MSB to LSB)
These bits contain the value of the watchdog counter. It is decremented every 12288 f_{MASTER} cycles (approximately). A reset is produced when it rolls over from 0x40 to 0x3F (T6 becomes cleared).

1. This bit is not used if the hardware watchdog option is enabled by option byte.

14.8.2 Window register (WWDG_WR)

Address offset: 0x01
Reset value: 0x7F

7	6	5	4	3	2	1	0
Reserved	W6	W5	W4	W3	W2	W1	W0
	rw	rw	rw	rw	rw	rw	rw

Bit 7 Reserved

Bits 6:0 **W[6:0]:** 7-bit window value
These bits contain the window value to be compared to the downcounter.

14.9 Window watchdog register map and reset values

Table 28. WWDG register map and reset values

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	WWDG_CR Reset value	WDGA 0	T6 1	T5 1	T4 1	T3 1	T2 1	T1 1	T0 1
0x01	WWDG_WR Reset value	- 0	W6 1	W5 1	W4 1	W3 1	W2 1	W1 1	W0 1

15 Independent watchdog (IWDG)

15.1 Introduction

The independent watchdog peripheral can be used to resolve processor malfunctions due to hardware or software failures. It is clocked by the 38 kHzLSI internal RC clock source, and thus stays active even if the main clock fails.

15.2 IWDG functional description

Figure 18 shows the functional blocks of the independent watchdog module.

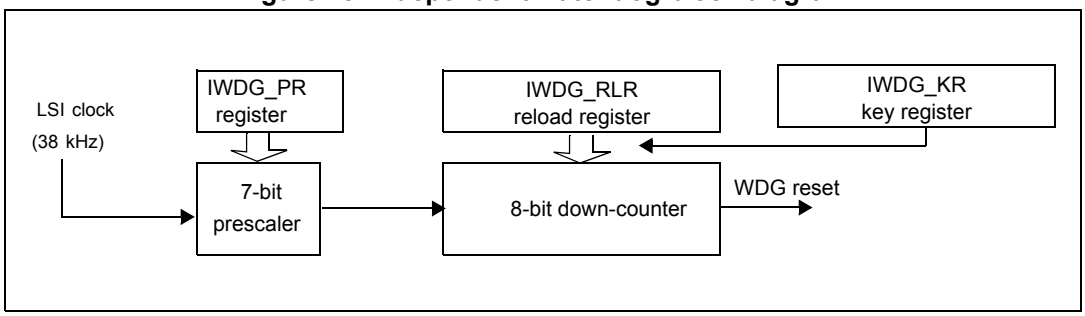
When the independent watchdog is started by writing the value 0xCC in the key register (IWDG_KR), the counter starts counting down from the reset value of 0xFF. When it reaches the end of count value (0x00) a reset signal is generated (IWDG RESET).

Once enabled, the independent watchdog can be configured through the IWDG_PR, and IWDG_RLR registers. The IWDG_PR register is used to select the prescaler divider feeding the counter clock. Whenever the KEY_REFRESH value (0xAA) is written in the IWDG_KR register, the IWDG is refreshed by reloading the IWDG_RLR value into the counter and the watchdog reset is prevented.

The IWDG_PR and IWDG_RLR registers are write protected. To modify them, first write the KEY_ACCESS code (0x55) in the IWDG_KR register. The sequence can be aborted by writing 0xAA in the IWDG_KR register to refresh it.

Refer to *Section 15.3: IWDG registers* for details on the IWDG registers.

Figure 18. Independent watchdog block diagram



Hardware watchdog feature

If the hardware watchdog feature has been enabled through the IWDG_HW option byte, the watchdog is automatically enabled at power-on, and generates a reset unless the key register is written by the software before the counter reaches end of count. Refer to the option byte description in the datasheet.

Timeout period

The maximum timeout period can be configured through the IWDG_PR and IWDG_RLR registers. It is determined by the following equation:

$$T = T_{LSI} \times P \times R$$

where:

T = Maximum timeout period

$T_{LSI} = 1/f_{LSI}$

$P = 2^{(PR[2:0] + 2)}$

R = RLR[7:0]+1

The IWDG counter must be refreshed by software before this timeout period expires. Otherwise, an IWDG reset will be generated after the following delay has elapsed since the last refresh operation:

$D = T + 3 \times T_{LSI}$

where D= delay between the last refresh operation and the IWDG reset.

Table 29. Min/Max IWDG timeout (LSI clock frequency = 38 kHz)

Prescaler divider	PR[2:0] bits	Timeout (ms)	
		RL[7:0]= 0x00	RL[7:0]= 0xFF
/4	0	0.11	26.95
/8	1	0.21	53.89
/16	2	0.42	107.79
/32	3	0.84	215.58
/64	4	1.68	431.16
/128	5	3.37	862.32
/256	6	6.74	1724.63

Using the IWDG in Halt/Active-halt mode

The IWDG can continue to work in Halt or Active-halt mode, depending on the configuration of the IWDG_HALT option byte. In this case, it can wake up the device from one of these modes. For more details, please refer to the Option Byte description in the datasheet.

Note: The application must configure correctly the IWDG timeout and refresh the IWDG counter before executing the HALT instruction, to avoid unexpected IWDG reset.

15.3 IWDG registers

15.3.1 Key register (IWDG_KR)

Address offset: 0x00

Reset value: 0xXX

7	6	5	4	3	2	1	0
KEY[7:0]							
w							

Bits 7:0 **KEY[7:0]**: Key value

The KEY_REFRESH value must be written by software at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0.

If the IWDG is not enabled by option byte (see datasheet for option byte description), the KEY_ENABLE value is the first value to be written in this register.

KEY_ENABLE value = 0xCC
Writing the KEY_ENABLE value starts the IWDG.
KEY_REFRESH value = 0xAA
Writing the KEY_REFRESH value refreshes the IWDG.
KEY_ACCESS value = 0x55
Writing the KEY_ACCESS value enables the access to the protected IWDG_PR and IWDG_RLR registers (see *Section 15.2*).

15.3.2 Prescaler register (IWDG_PR)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved					PR[2:0]		
					rw		

Bits 7:3 Reserved

Bits 2:0 **PR[2:0]:** Prescaler divider

These bits are write access protected (see *Section 15.2*). They can be written by software to select the prescaler divider feeding the counter clock.

000: divider /4
001: divider /8
010: divider /16
011: divider /32
100: divider /64
101: divider /128
110: divider /256
111: Reserved

15.3.3 Reload register (IWDG_RLR)

Address offset: 0x02

Reset value: 0xFF

7	6	5	4	3	2	1	0
RL[7:0]							
rw							

Bits 7:0 **RL[7:0]**: Watchdog counter reload value

These bits are write access protected (see *Section 15.2*). They are written by software to define the value to be loaded in the watchdog counter each time the value 0xAA is written in the IWDG_KR register. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to *Table 29*.

15.3.4 IWDG register map and reset values

Table 30. IWDG register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	IWDG_KR Reset value	KEY[7:0] xxxxxxxx							
0x01	IWDG_PR Reset value	- 0	- 0	- 0	- 0	- 0	PR2[2:0] 000		
0x02	IWDG_RLR Reset value	RL7[7:0] 11111111							

16 Timer overview

The microcontroller has two types of TIM timers, two general purpose (TIM2/ TIM3), and one basic timer (TIM4). They have different features but are based on a common architecture. This makes it easier to design applications using the various timers (identical register mapping, common basic features).

Although the timers do not share any resources, they can be linked together and synchronized.

This section gives a comparison of the different timer features and glossary of internal timer signal names.

Section 17: 16-bit general purpose timer (TIM2/TIM3) contains a full description of all the various timer modes.

Table 31. Timer characteristics

Symbol	Parameter	Min.	Typ.	Max.	Unit
$t_{w(ICAP)in}$	Input capture pulse time	2			t_{MASTER}
$t_{res(TIM)}$	Timer resolution time	1			t_{MASTER}
Res_{TIM}	Timer resolution with 16-bit counter		16		bit
	Timer resolution with 8-bit counter		8		bit
$t_{COUNTER}$	Counter clock period when internal clock is selected		1		t_{MASTER}
t_{MAX_COUNT}	Maximum possible count with 16-bit counter			65,536	t_{MASTER}
	Maximum possible count with 8-bit counter			256	t_{MASTER}

16.1 Timer feature comparison

Table 32. Timer feature comparison

Timer	Counter resol-ution	Counter type	Prescaler factor	Capture/ compare chan-nels	Comple-mentary outputs	Repet-ition counter	External trigger input	External break input	Timer synchr-onization/ chaining
TIM2 & TIM3 (general purpose timers)	16-bit	Up/down	Any power of 2 from 1 to 128	2	None	No	1	1	Yes
TIM4 (basic timer)	8-bit	Up	Any power of 2 from 1 to 32768	0			0	0	

16.2 Glossary of timer signal names

Table 33. Glossary of internal timer signals

Internal signal name	Description	Related figures
BI	Break interrupt	*Figure 19: TIMx general block diagram on page 106*
CC*i*I, CC1I, CC2I, CC3I, CC4I	Capture/compare interrupt	
CK_CNT	Counter clock	*Figure 23: Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2 on page 110*
CK_PSC	Prescaler clock	
CNT_EN	Counter enable	
CNT_INIT	Counter initialize	*Figure 32: TI2 external clock connection example on page 116*
ETR	External trigger from TIMx_ETR pin	*Figure 34: External trigger input block on page 117*
ETRF	External trigger filtered	
ETRP	External trigger prescaled	
f_{MASTER}	Timer peripheral clock from clock controller (CLK)	*Figure 12: Clock structure on page 61*
IC*i*, IC1, IC2	Input capture	*Figure 51: Input stage of TIM 1 channel 1 on page 130*
IC*i*PS, IC1PS, IC2PS	Input capture prescaled	
ITRx, ITR1, ITR2, ITR3	Internal trigger input tied to TRGO of other TIM timers	*Figure 19: TIMx general block diagram on page 106*
MATCH1	Compare match	*Figure 41: Trigger/master mode selection blocks on page 122* and *Section 17.7.2: Control register 2 (TIMx_CR2) on page 145*
OC*i*, OC1, OC2	Timer output channel	*Figure 17.5.5: Forced output mode on page 133*
OC*i*REF, OC1REF, OC2REF	Output compare reference signal	
TGI	Trigger interrupt	*Figure 30: Clock/trigger controller block diagram on page 115*
TI*i*, TI1, TI2	Timer input	*Figure 51: Input stage of TIM 1 channel 1 on page 130*
TI*i*F, TI1F, TI2F	Timer input filtered	
TI1F_ED	Timer input filtered edge detector	
TI*i*FPx, TI1FP1, TI1FP2, TI2FP1, TI2FP2	Timer input filtered prescaled	
TRC	Trigger capture	
TRGI	Trigger input to clock/trigger/slave mode controller	*Figure 31: Control circuit in normal mode, fMASTER divided by 1 on page 116*
TRGO	Trigger output tied to trigger input ITRx of other timers	*Figure 19: TIMx general block diagram on page 106*

Table 33. Glossary of internal timer signals (continued)

Internal signal name	Description	Related figures
UEV	Update event	*Figure 23: Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2 on page 110*
UIF	Update interrupt	

17 16-bit general purpose timer (TIM2/TIM3)

17.1 Introduction

This chapter describes TIM2 and TIM3 which are identical timers and referred to as TIMx.

Each general purpose timer (TIMx) has a 16-bit up-down auto-reload counter driven by a programmable prescaler.

In this section, the index *i*, may be 1 or 2 referring to the two capture/compare channels.

The timers may be used for a variety of purposes, including:

- Time base generation
- Measuring the pulse lengths of input signals (input capture)
- Generating output waveforms (output compare, PWM and One Pulse Mode)
- Interrupt capability on various events (capture, compare, overflow, break, trigger)
- Synchronization with other timers or external signals (external clock, reset, trigger and enable)

These timers are ideally suited for a wide range of control applications, including those requiring center-aligned PWM capability.

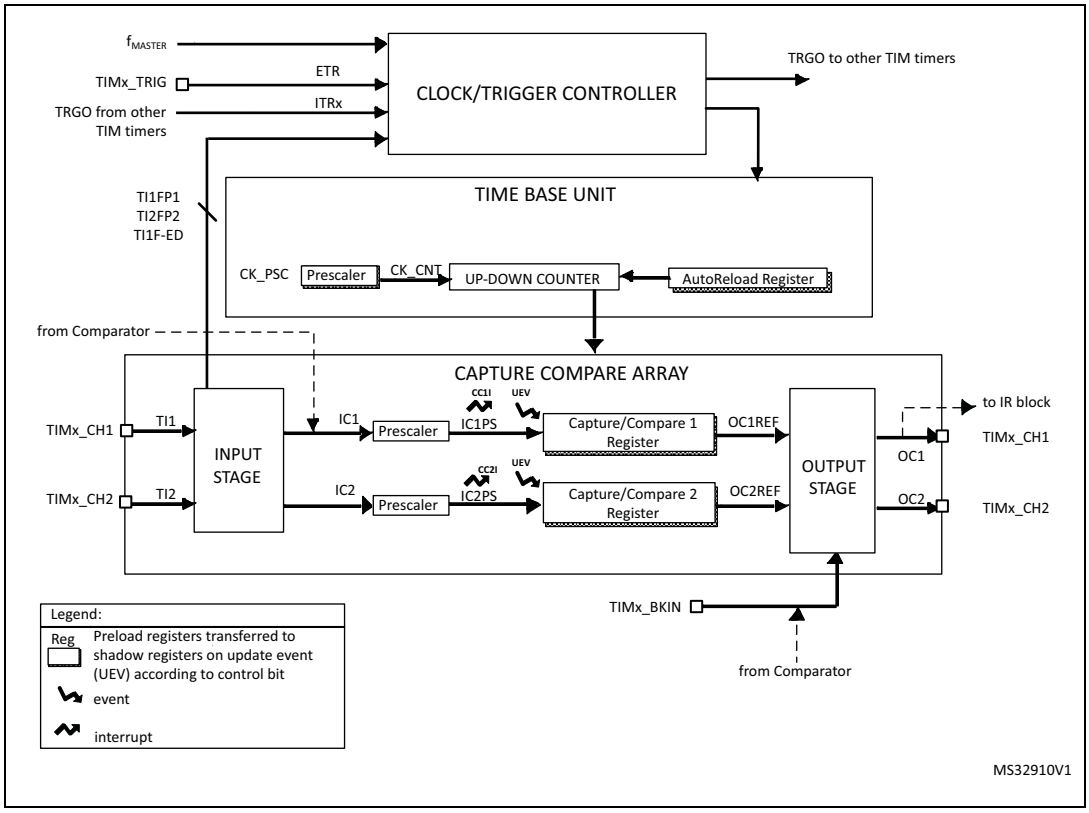
The timer clock can be sourced from internal clocks or from an external source selectable through a configuration register.

17.2 TIMx main features

TIMx features include:

- 16-bit up, down, up/down counter auto-reload counter.
- 3-bit programmable prescaler allowing the counter clock frequency to be divided “on the fly” by any power of 2 between 1 and 128.
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- 2 independent channels that can alternately be configured as:
  - Input capture
  - Output compare
  - PWM generation (edge and center-aligned mode)
  - One Pulse Mode output
- Break input to put the timer output signals in reset state or in a known state.
- Interrupt 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

Figure 19. TIMx general block diagram

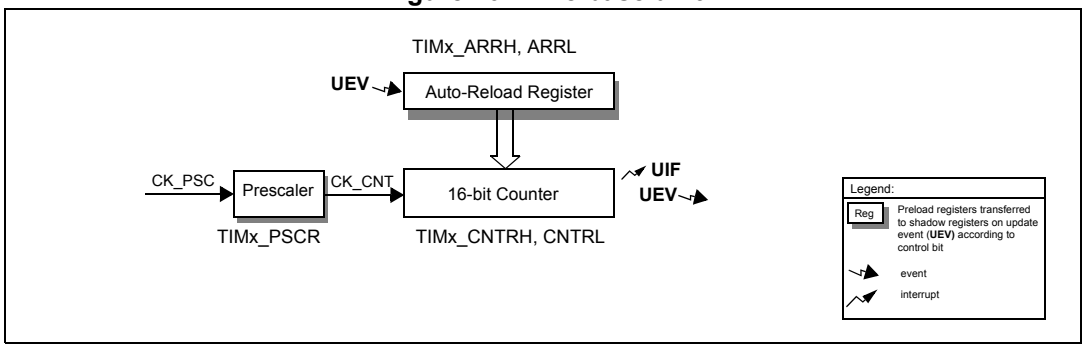


17.3 TIMx time base unit

The timer has a *Time base unit* that includes:

- 16-bit up/down counter
- 16-bit auto-reload register
- Prescaler

Figure 20. Time base unit



The 16-bit counter, the prescaler and the auto-reload register can be written or read by software.

The auto-reload register is composed of a preload register plus a shadow register.

Writing to the auto-reload register can be done in two modes:

- **Auto-reload preload enabled** (ARPE bit set in the TIMx_CR1 register). In this mode, when data is written to the autoreload register, it is kept in the preload register and transferred into the shadow register at the next update event (UEV).
- **Auto-reload preload disabled** (ARPE bit cleared in the TIMx_CR1 register). In this mode, when data is written to the autoreload register it is transferred into the shadow register immediately.

An update event is generated:

- On a counter overflow or underflow.
- By software, setting the UG bit in the TIMx_EGR register.
- By an trigger event from the clock/trigger controller.

With preload enabled (ARPE=1), when an update event occurs: the auto-reload shadow register is updated with the preload value (TIMx_ARR) and the buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register).

The update event (UEV) can be disabled by setting the UDIS bit in the TIMx_CR1

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.

Note: *Note that the actual counter enable signal CNT_EN is set 1 clock cycle after CEN.*

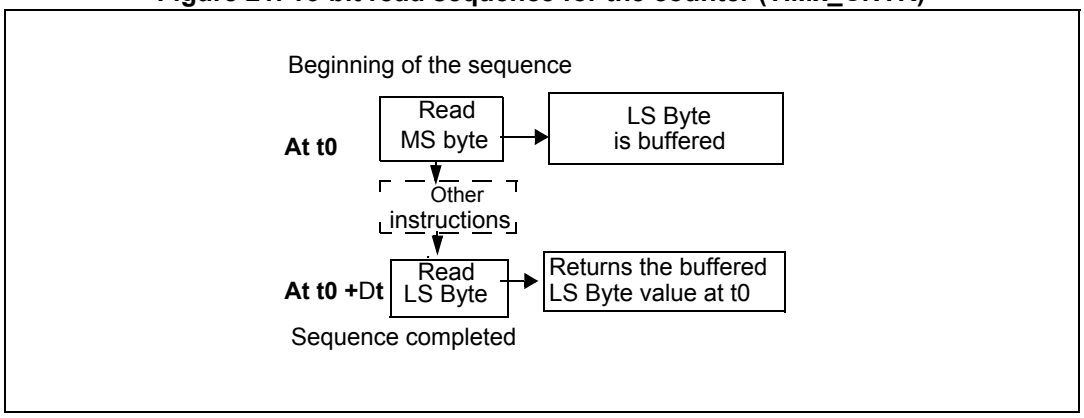
17.3.1 Reading and writing to the 16-bit counter

There is no buffering when writing the counter. Both TIMx_CNTRH and TIMx_CNTRL can be written at any time, so it is suggested not to write a new value into the counter while it is running to avoid loading a wrong intermediate content.

An 8-bit buffer is implemented for the read. The user must read the MS byte first, then the LS byte value is buffered automatically, as described in *Figure 21*. This buffered value remains unchanged until the 16-bit read sequence is completed.

Note: Do not use the LDW instruction to read the 16-bit counter, because it reads the LS byte first, and would return a wrong result.

Figure 21. 16-bit read sequence for the counter (TIMx_CNTR)



17.3.2 Write sequence for 16-bit TIMx_ARR register

16-bit values are loaded in the TIMx_ARR register through preload registers. This must be performed by two write instructions, one for each byte. The MS byte must be written first.

The shadow register update is blocked as soon as the MS byte has been written, and stays blocked until the LS byte has been written. Do not use the LDW instruction, as this writes the LS byte first, and would produce wrong results in this case.

17.3.3 Prescaler

The prescaler is based on a 7-bit counter controlled through a 3-bit register (in TIMx_PSCR register). It can be changed on the fly as this control register is buffered. It can divide the counter clock frequency by 1, 2, 4, 8, 16, 32, 64 or 128.

The counter clock frequency is calculated as follows:

$$f_{CK_CNT} = f_{CK_PSC}/2^{(PSCR[2:0])}$$

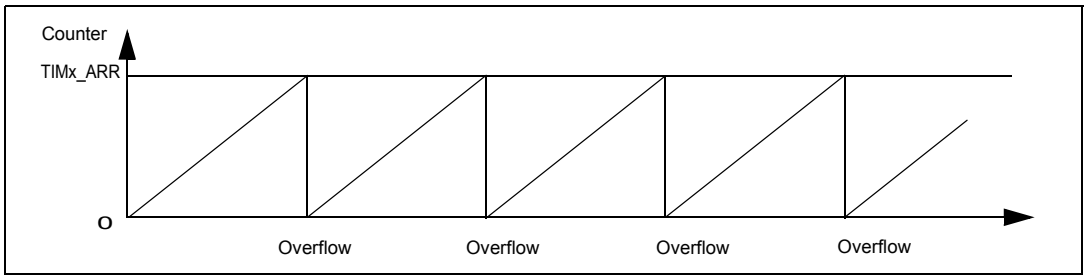
The new prescaler value is taken into account in the following period (after the next counter update event).

17.3.4 Up-counting mode

In up-counting mode, the counter counts from 0 to a user-defined compare value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event, and an update event (UEV) if the UDIS bit is 0 in the TIMx_CR1 register.

Figure 22 shows an example of this counting mode.

Figure 22. Counter in up-counting mode



An update event can also be generated by setting the bit UG in the TIMx_EGR register (by software or by using the trigger controller).

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 will occur until UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescaler division factor 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 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_SR1 register) is set (depending on the URS bit):

The auto-reload shadow register is updated with the preload value (TIMx_ARR),

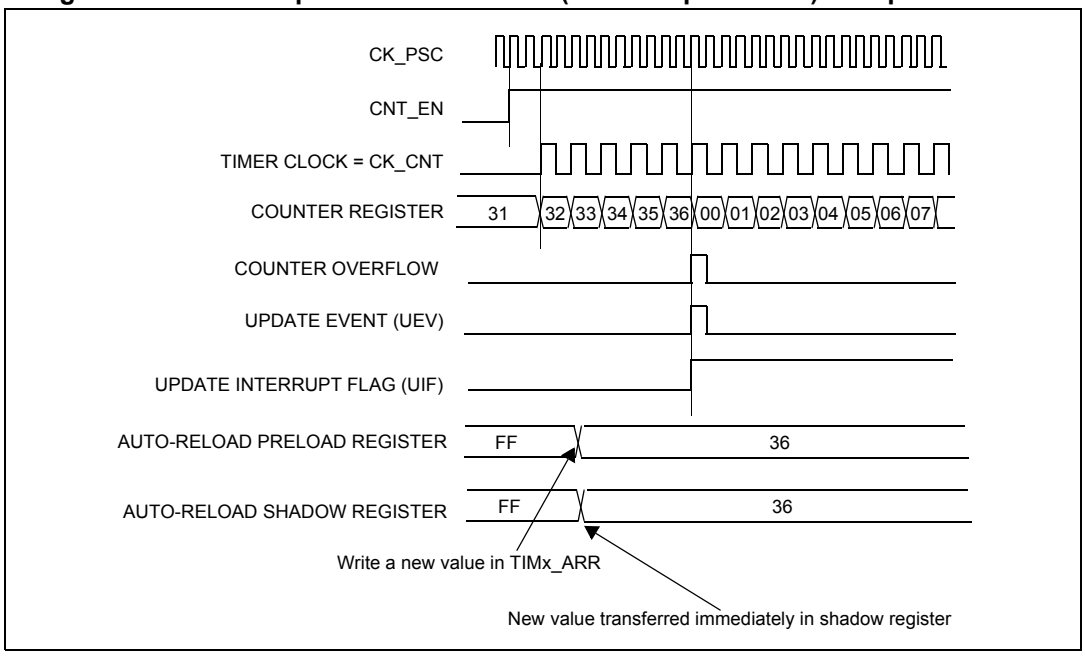
The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register).

The following figures show two examples of the counter behavior for different clock frequencies when TIMx_ARR=36h.

In *Figure 23* the prescaler divider is set to 2, so the counter clock (CK_CNT) frequency is at half the frequency of the the prescaler clock source (CK_PSC).

In *Figure 23* the autoreload preload is disabled (ARPE=0), so the shadow register is changed immediately and counter overflow occurs when upcounting reaches 36h. This generates an update event.

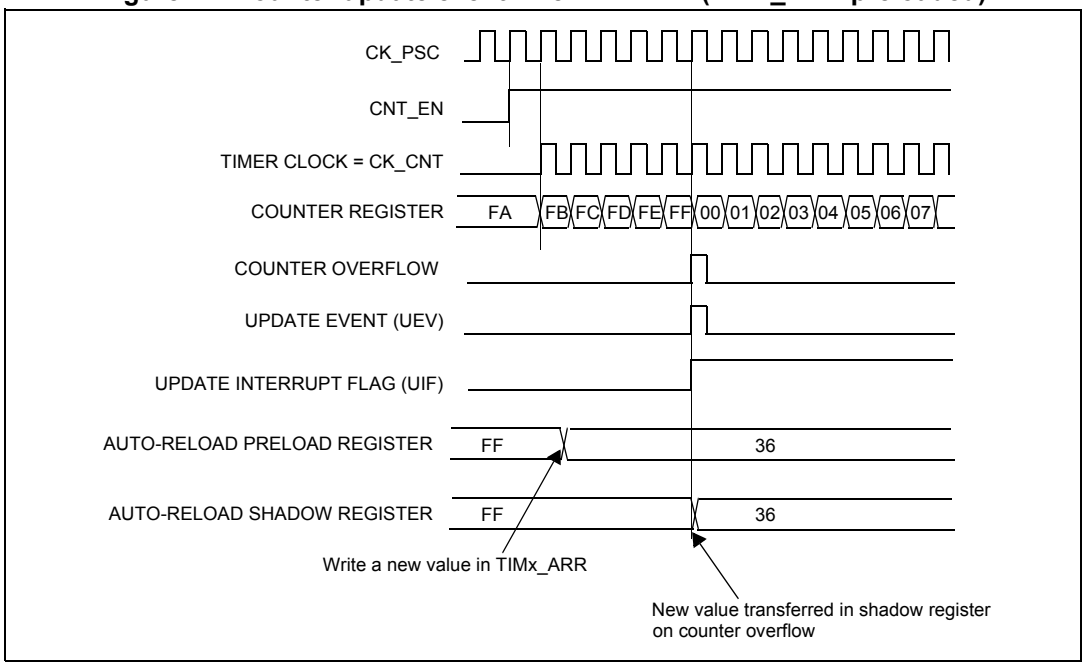
Figure 23. Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2



In *Figure 24* the prescaler divider is set to 1, so CK_CNT has the same frequency as CK_PSC.

In *Figure 24* autoreload preload is enabled (ARPE=1), so the next counter overflow occurs at FFh. The new autoreload value register value of 36h is taken into account after the overflow which generates an update event.

Figure 24. Counter update event when ARPE=1 (TIMx_ARR preloaded)

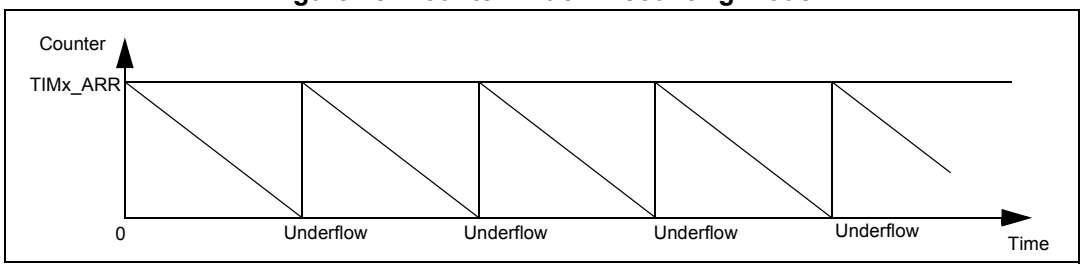


17.3.5 Down-counting mode

In down-counting 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 and an update event (UEV) if the UDIS bit is 0 in the TIMx_CR1 register.

Figure 25 shows an example of this counting mode.

Figure 25. Counter in down-counting mode



An update event can also be generated by setting the bit UG in the TIMx_EGR register (by software or by using the clock/trigger 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 will occur 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 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_SR1 register) is set (depending on the URS bit):

The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register),

The auto-reload shadow 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=36h.

In downcounting mode, preload is normally not used so that the new value is taken into account in the next period (see *Figure 26*).

Figure 26. Counter update when ARPE=0 (ARR not preloaded) with prescaler = 2

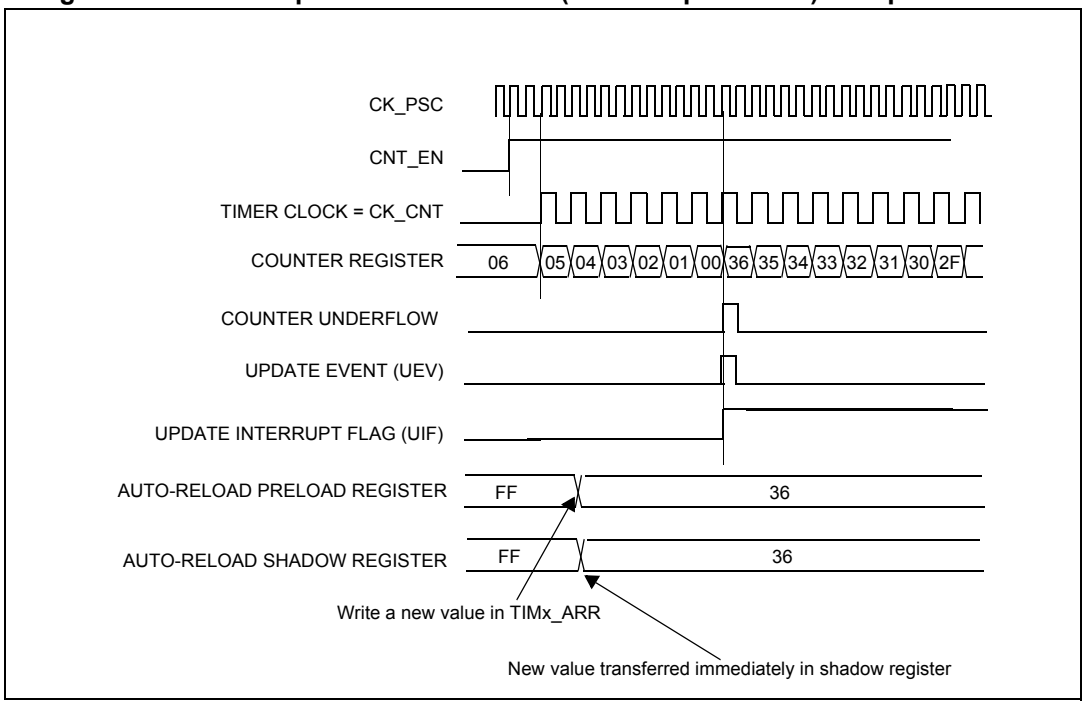
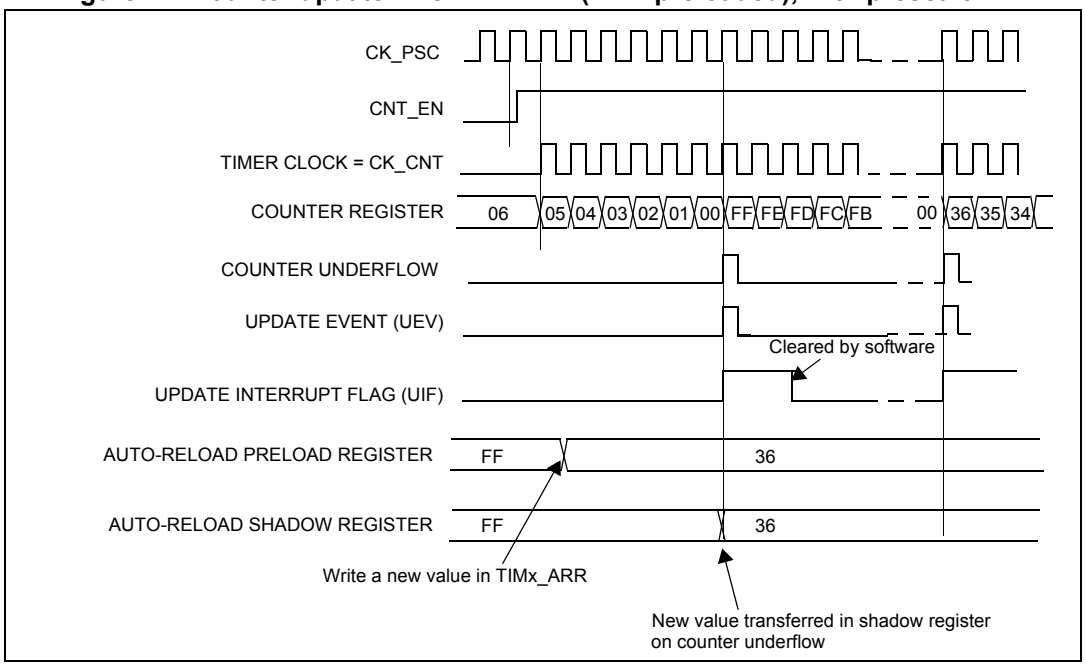


Figure 27. Counter update when ARPE=1 (ARR preloaded), with prescaler = 1



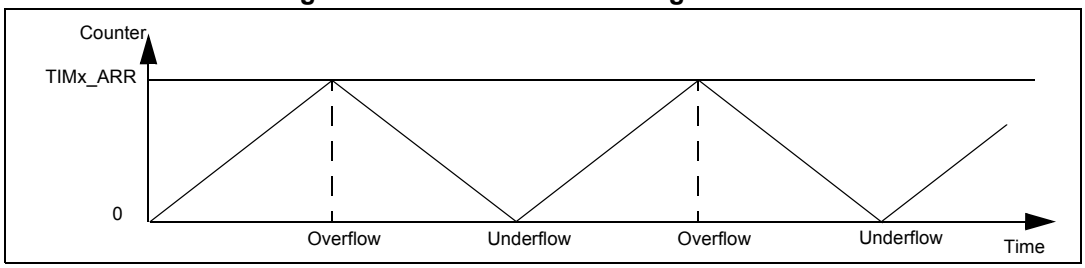
17.3.6 Center-aligned mode (up/down counting)

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 down to 0 and generates a counter underflow event. Then it restarts counting from 0.

In this mode, the DIR direction bit in the TIMx_CR1 register cannot be written. It is updated by hardware and gives the current direction of the counter.

The *Figure 28* shows an example of this counting mode.

Figure 28. Counter in center-aligned mode



The update event is generated at each counter overflow and at each counter underflow.

Setting the bit UG in the TIMx_EGR register (by software or by using the clock/trigger 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 update event (UEV) can be disabled by software 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 will occur until UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

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 request will be sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

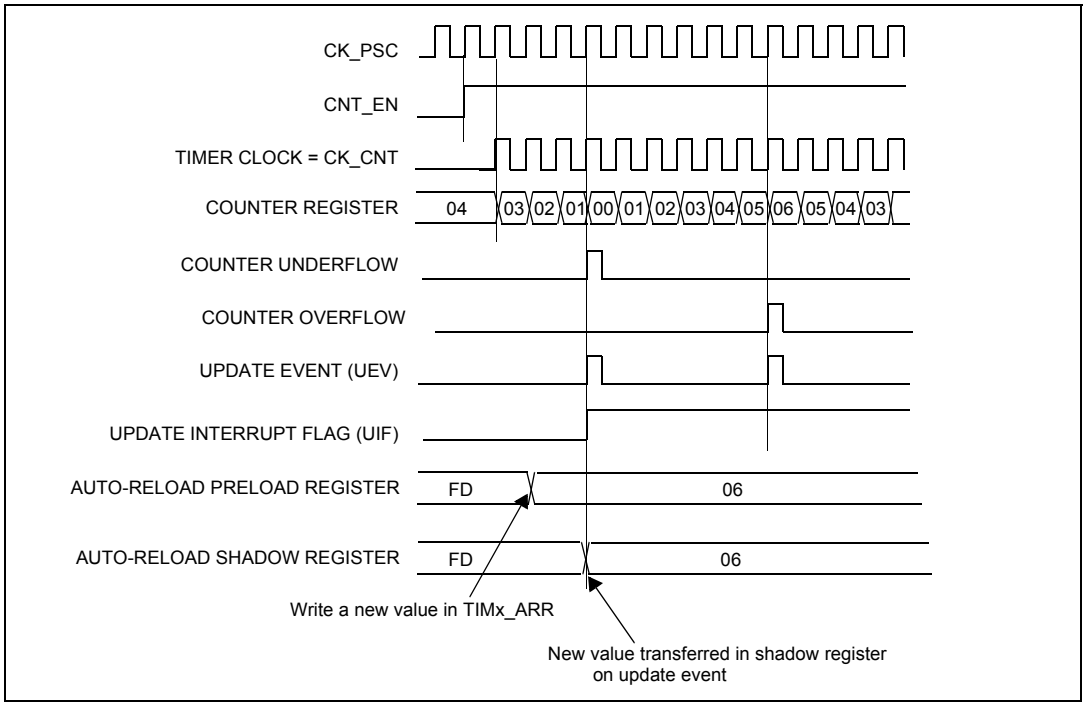
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR1 register) is set (depending on the URS bit).

The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSCR register).

The auto-reload shadow 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).

Hereafter are some examples of the counter behavior for different clock frequencies.

Figure 29. Counter timing diagram, CK_PSC divided by 1, TIMx_ARR=06h, ARPE=1



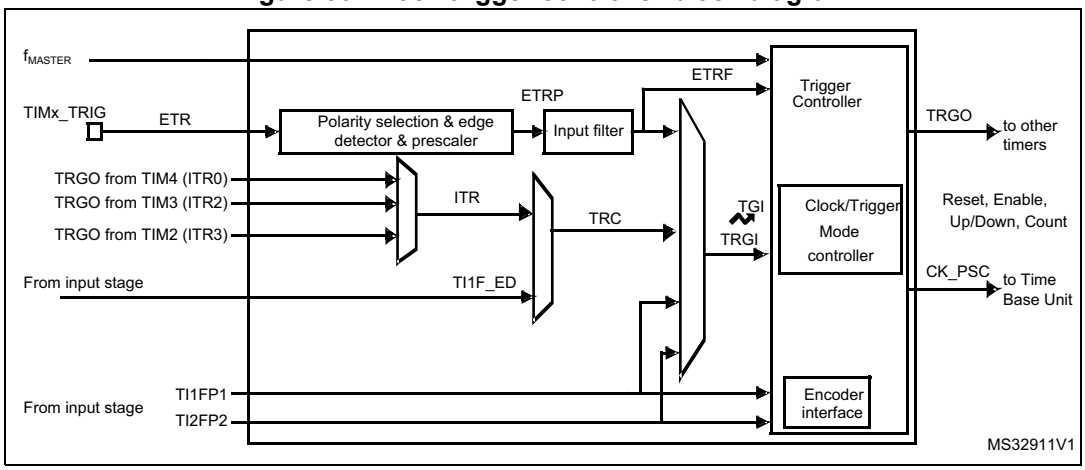
Hints on using center-aligned mode:

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter will start counting up or down depending on the value written in the DIR bit in the TIMx_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if you write a value in the counter that is greater than the auto-reload value (TIMx_CNT>TIMx_ARR). For example, if the counter was counting up, it will continue to count up.
  - The direction is updated if you write 0 or write the TIMx_ARR value in the counter but no Update Event UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIMx_EGR register) just before starting the counter and not to write the counter while it is running.

17.4 TIMx clock/trigger controller

The clock/trigger controller allows you to configure the timer clock sources, input triggers and output triggers. The block diagram is shown in *Figure 30.*

Figure 30. Clock/trigger controller block diagram



17.4.1 Prescaler clock (CK_PSC)

The Time base unit prescaler clock (CK_PSC) can be provided by the following clock sources:

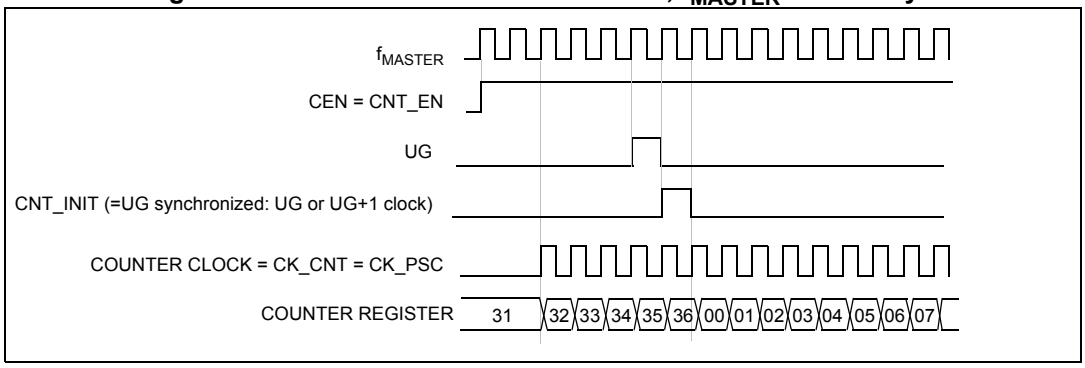
- Internal clock (f_{MASTER})
- External clock mode 1: external timer input (TIx)
- External clock mode 2: external trigger input ETR
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer. Refer to *Using one timer as prescaler for another timer on page 123* for more details.

17.4.2 Internal clock source

If both the clock/trigger mode controller and the external trigger input are disabled (SMS=0b000 in TIMx_SMCR and ECE=0 in the TIMx_ETR register), then the CEN, DIR and UG bits 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.

Figure 31 shows the behavior of the control circuit and the up-counter in normal mode, without prescaler.

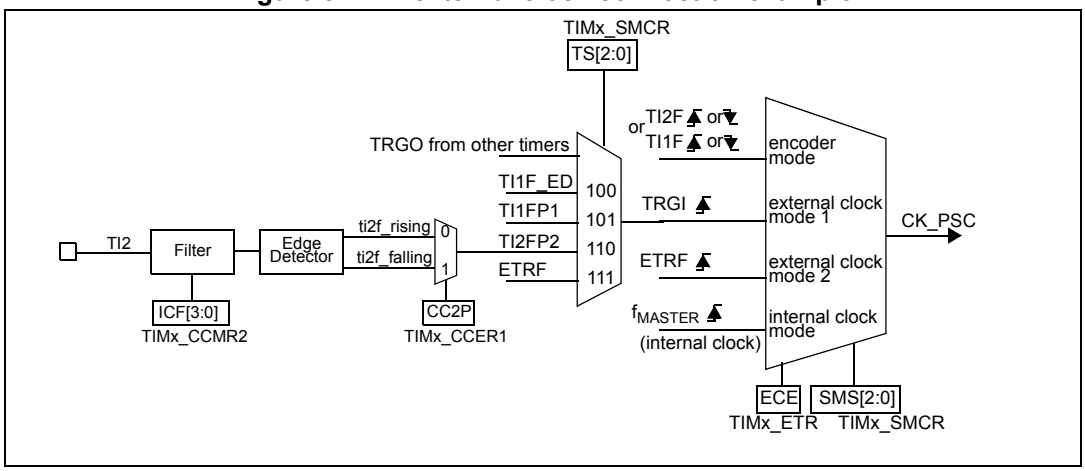
Figure 31. Control circuit in normal mode, f_{MASTER} divided by 1



17.4.3 External clock source mode 1

The counter can count at each rising or falling edge on a selected timer input. This mode is selected when SMS=0b111 in the TIMx_SMCR register.

Figure 32. TI2 external clock connection example



For example, to configure the up-counter 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_CCMR2 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR2 register (if no filter is needed , keep IC2F=0000).

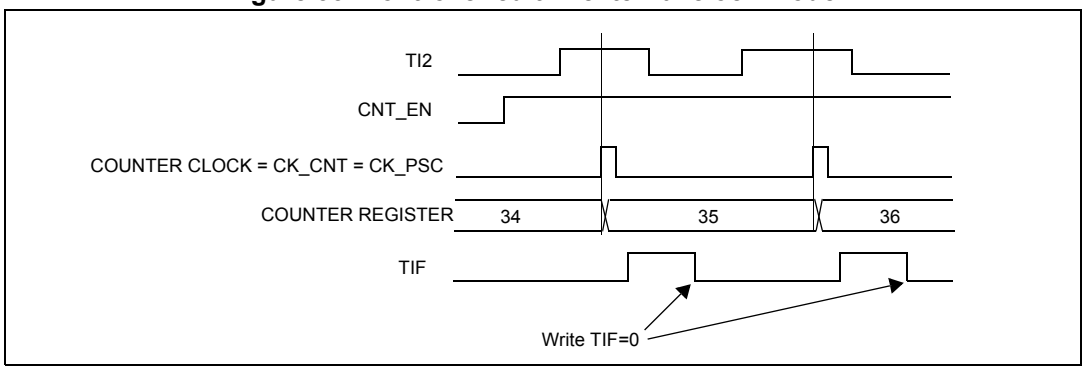
Note: The capture prescaler is not used for triggering, so you don’t need to configure it. Also you don’t need to configure the TI2S bits, they only select the input capture source.

3. Select rising edge polarity by writing CC2P=0 in the TIMx_CCER1 register.
4. Configure the timer in external clock mode 1 by writing SMS=0b111 in the TIMx_SMCR register.
5. Select TI2 as the input source by writing TS=110 in the TIMx_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

When a rising edge occurs on TI2, the counter counts once and the trigger flag is set (TIF bit in the TIMx_SR1 register) and an interrupt request can be sent if enabled (depending on the TIE in TIMx_IER register).

The delay between the rising edge on TI2 and the actual reset of the counter is due to the resynchronization circuit on TI2 input.

Figure 33. Control circuit in external clock mode 1

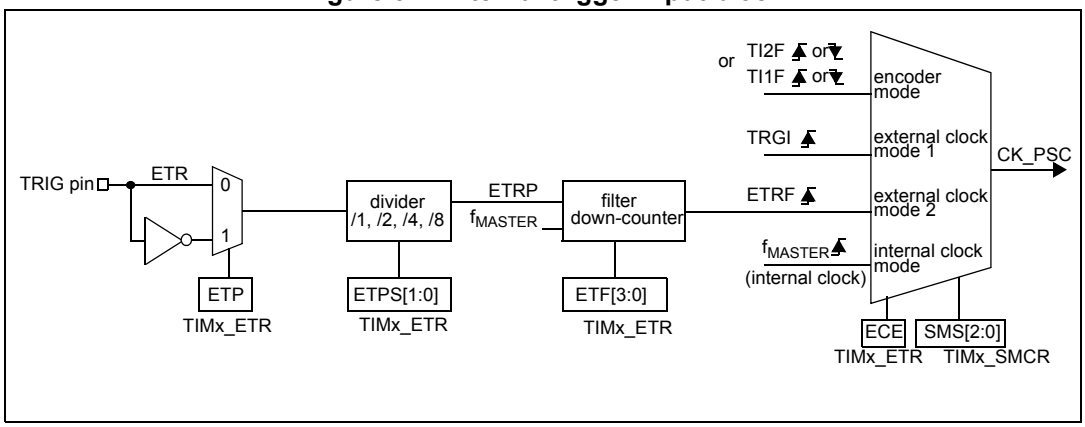


17.4.4 External clock source mode 2

The counter can count at each rising or falling edge on the external trigger input ETR. This mode is selected by writing ECE=1 in the TIMx_ETR register.

The *Figure 34* gives an overview of the external trigger input block.

Figure 34. External trigger input block



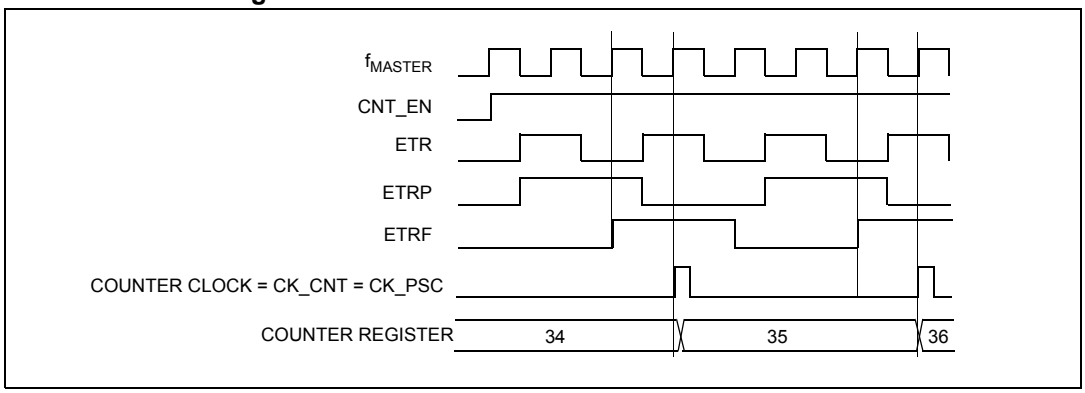
For example, to configure the up-counter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0b0000 in the TIMx_ETR register.
2. Set the prescaler by writing ETPS[1:0]=0b01 in the TIMx_ETR register
3. Select rising edge detection on the TRIG pin by writing ETP=0 in the TIMx_ETR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx_ETR 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 reset of the counter is due to the resynchronization circuit on the ETRP signal.

Figure 35. Control circuit in external clock mode 2



17.4.5 Trigger synchronization

There are four trigger inputs (refer to *Table 33: Glossary of internal timer signals on page 103*):

- ETR
- TI1
- TI2
- TRGO from other timers

The TIMx timer can be synchronized with an external trigger in three modes: trigger standard mode, trigger reset mode and trigger gated mode.

Trigger standard mode

The counter can start in response to an event on a selected input.

In the following example, the up-counter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. TI2S bits are selecting the input capture source only, and don't need to be configured too. Write CC2P=0 in TIMx_CCER1 register to select rising edge polarity.
- Configure the timer in trigger mode by writing SMS=0b110 in the TIMx_SMCR register. Select TI2 as the input source by writing TS=0b110 in the TIMx_SMCR register.

When a rising edge occurs on TI2, the counter starts counting on the internal clock and the TIF flag is set.

The delay between the rising edge on TI2 and the actual reset of the counter is due to the resynchronization circuit on TI2 input.

Figure 36. Control circuit in trigger mode

Trigger reset mode

The counter and its prescaler can be re-initialized 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.

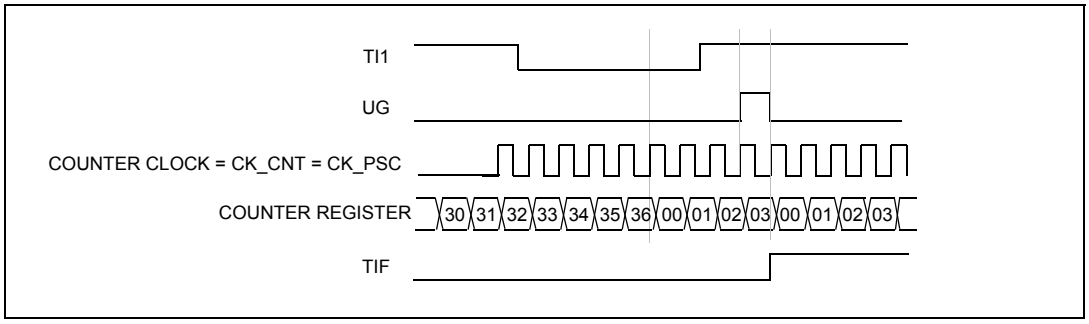
In the following example, the up-counter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, and do not need to be configured either. Write CC1P=0 in TIMx_CCER1 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=0b101 in TIMx_SMCR register.
- Start the counter by writing CEN=1 in the TIMx_CR1 register.

The counter starts counting on the internal clock, then behaves normally until TI1 rising edge. When TI1 rises, the counter is cleared and restarts from 0. In the meantime, the trigger flag is set (TIF bit in the TIMx_SR1 register) and an interrupt request can be sent if enabled (depending on the TIE in TIMx_IER register).

The following figure shows this behaviour when the auto-reload register TIMx_ARR=36h. 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 37. Control circuit in trigger reset mode



Trigger gated mode

The counter can be enabled depending on the level of a selected input.

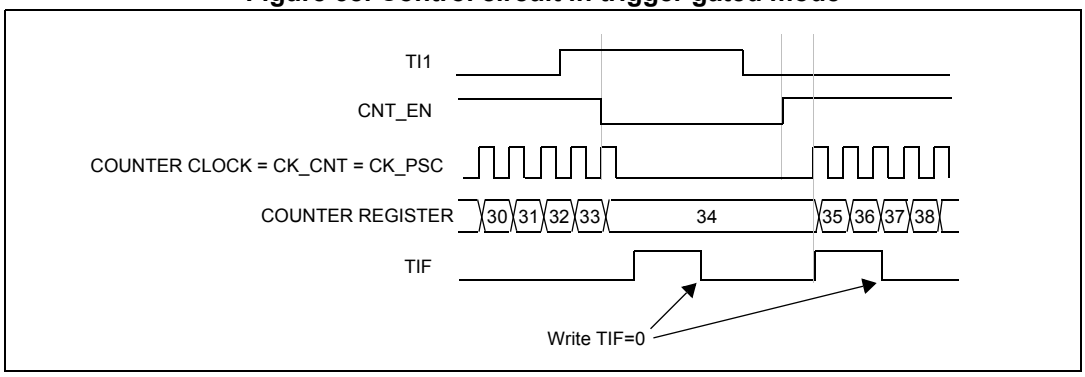
In the following example, the up-counter counts only when TI1 input is low:

1. Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, and do not need to be configured either. Write CC1P=1 in TIMx_CCER1 register to validate the polarity (and detect low level only).
2. Configure the timer in trigger gated mode by writing SMS=0b101 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
3. Enable the counter by writing CEN=1 in the TIMx_CR1 register (in trigger gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

The counter starts counting on the internal clock as long as TI1 is low and stops as soon as TI1 becomes high. The TIF flag is set both when the counter starts or stops.

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 38. Control circuit in trigger gated mode



Combining trigger modes with external clock mode 2

The external clock mode 2 can be used in addition to another trigger mode. In this case, ETR is used as external clock input, and another input can be selected as trigger input (in trigger standard mode, trigger reset mode or trigger gated mode). Take care that you must not select ETR as TRGI (through the TS bits in TIMx_SMCR register).

In the following example, the up-counter counts at each rising edge on ETR as soon as a rising edge has occured on TI1 (standard trigger mode with external ETR clock):

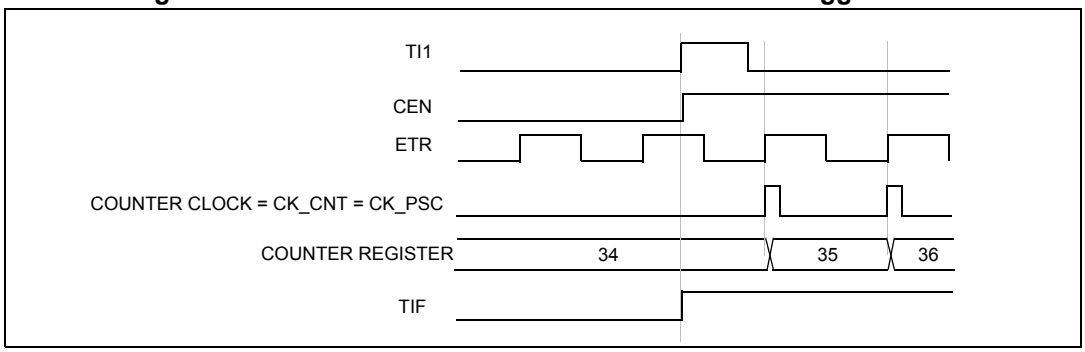
- Configure the external trigger input circuit by writing the TIMx_ETR register. In this example, we don't need any filter and write ETF=0b0000. Write ETPS=00 to disable the prescaler, ETP=0 to detect rising edges on ETR and ECE=1 to enable the external clock mode 2.
- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0b0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, and do not need to be configured either. Write CC1P=0 in TIMx_CCER1 register to select rising edge polarity.

- Configure the timer in trigger mode by writing SMS=0b110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=0b101 in TIMx_SMCR register.

A rising edge on TI1 enables the counter and sets the TIF flag. Then the counter counts on ETR rising edges.

The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input. The delay between the rising edge on ETR and the actual reset of the counter is due to the resynchronization circuit on the ETRP signal.

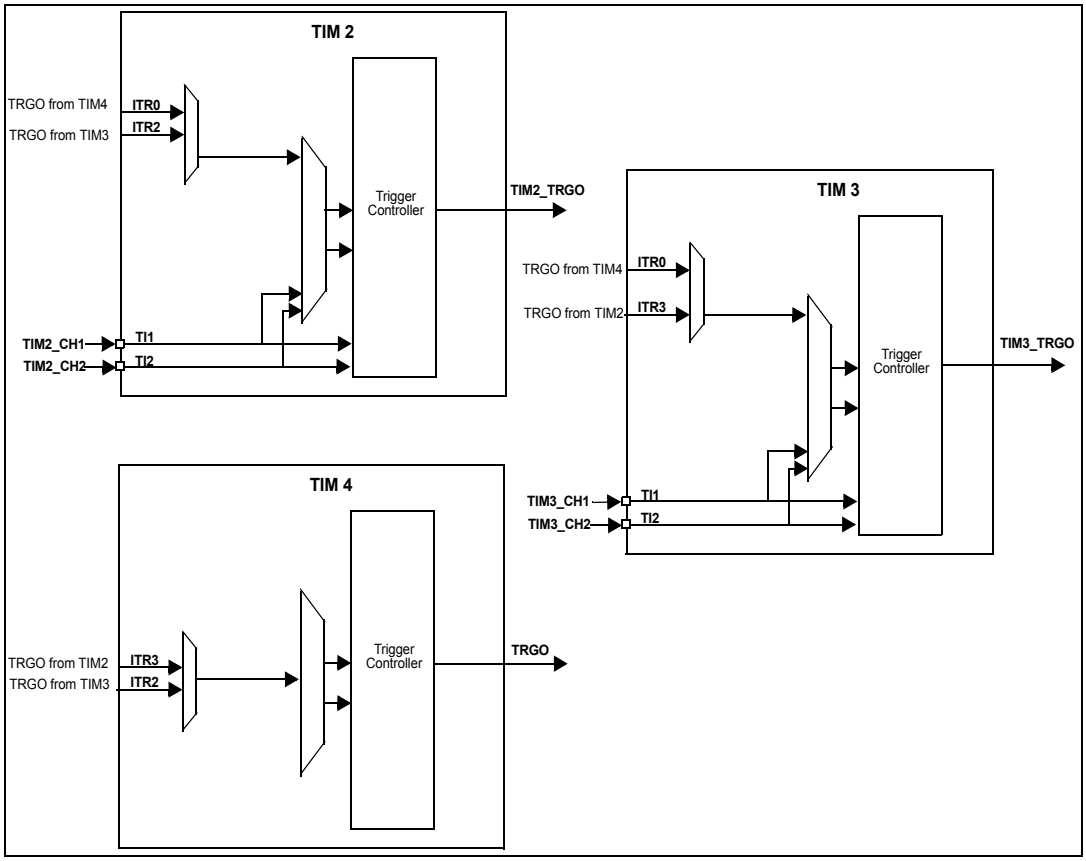
Figure 39. Control circuit in external clock mode 2 + trigger mode



17.4.6 Synchronization from other timers

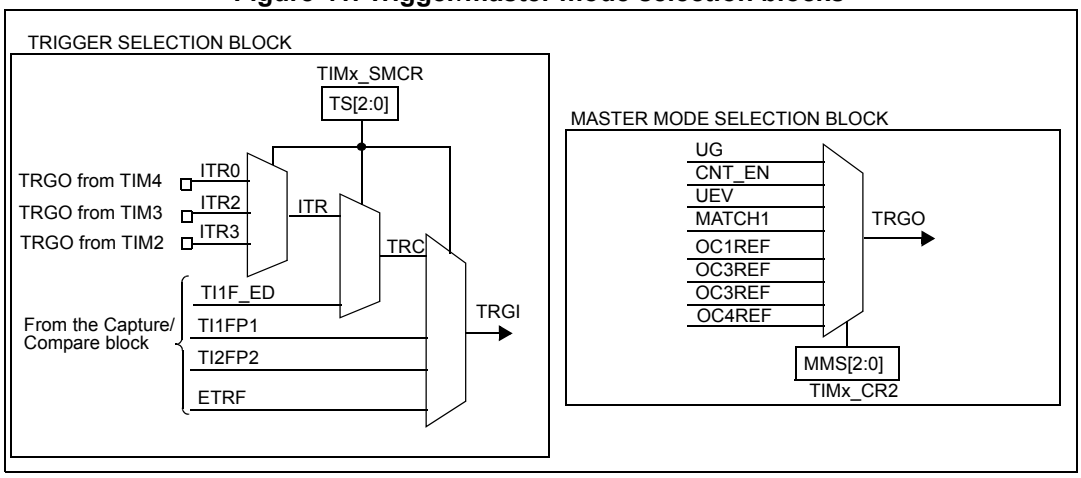
The timers are linked together internally for timer synchronization or chaining. When one timer is configured in master mode, it can output a trigger (TRGO) to reset, start, stop or clock the counter of any other Timer configured in slave mode.

Figure 40. Timer chaining system implementation example



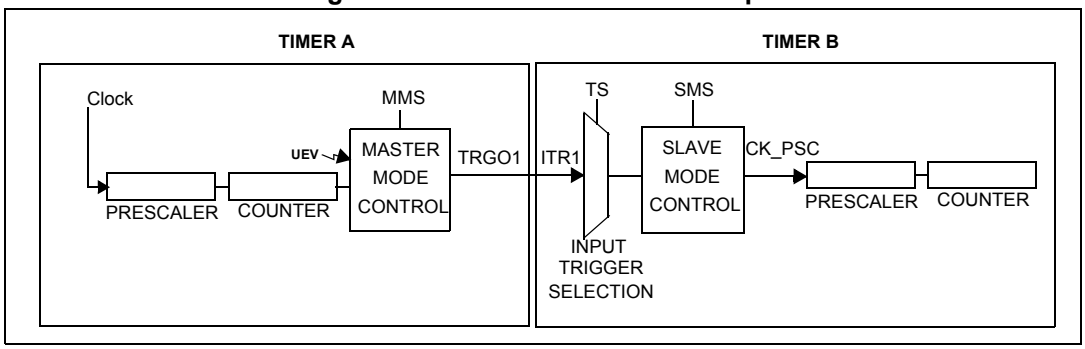
The following figure presents an overview of the trigger selection and the master mode selection blocks.

Figure 41. Trigger/master mode selection blocks



Using one timer as prescaler for another timer

Figure 42. Master/slave timer example



For example, you can configure Timer A to act as a prescaler for Timer B. Refer to *Figure 43*. To do this:

1. Configure Timer A in master mode so that it outputs a periodic trigger signal on each update event UEV. To configure that a rising edge is output on TRGO1 each time an update event is generated, write MMS=010 in the TIMx_CR2 register,.
2. Connect the TRGO1 output of Timer A to Timer B, Timer B must be configured in slave mode using ITR1 as internal trigger. Select this through the TS bits in the TIMx_SMCR register (writing TS=001).
3. Put the clock/trigger controller in external clock mode 1, by writing SMS=111 in the TIMx_SMCR register. This causes Timer B to be clocked by the rising edge of the periodic Timer A trigger signal (which corresponds to the Timer A counter overflow).
4. Finally enable both timers by setting their respective CEN bits (TIMx_CR1 register).

Note: If OCi is selected on Timer A as trigger output (MMS=1xx), its rising edge is used to clock the counter of Timer B.

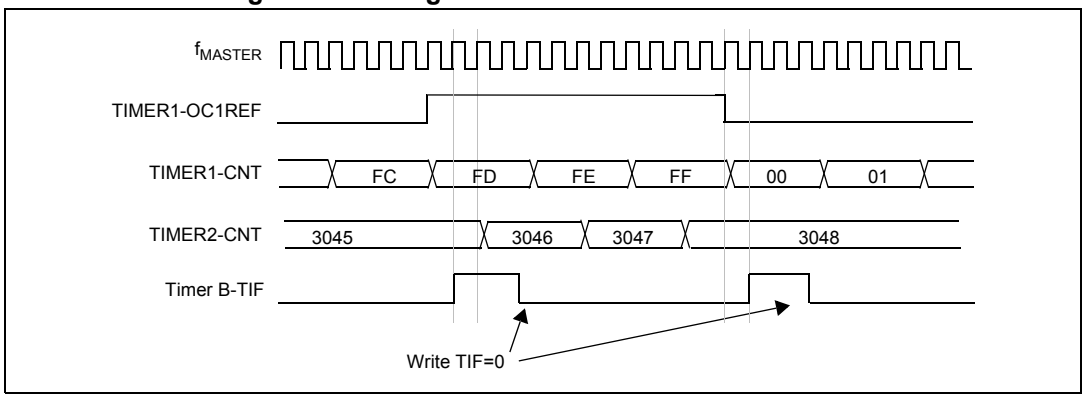
Using one timer to enable another timer

In this example, we control the enable of Timer B with the output compare 1 of Timer A. Refer to *Figure 43* for connections. Timer B counts on the divided internal clock only when OC1REF of Timer A is high. Both counter clock frequencies are divided by 4 by the prescaler compared to f_{MASTER} ($f_{CK_CNT} = f_{MASTER}/4$).

1. Configure Timer A master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIMx_CR2 register).
2. Configure the Timer A OC1REF waveform (TIMx_CCMR1 register).
3. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).
4. Configure Timer B in trigger gated mode (SMS=101 in TIMx_SMCR register).
5. Enable Timer B by writing ‘1’ in the CEN bit (TIMx_CR1 register).
6. Start Timer A by writing ‘1’ in the CEN bit (TIMx_CR1 register).

Note: The counter 2 clock is not synchronized with counter 1, this mode only affects the Timer B counter enable signal.

Figure 43. Gating Timer B with OC1REF of Timer A

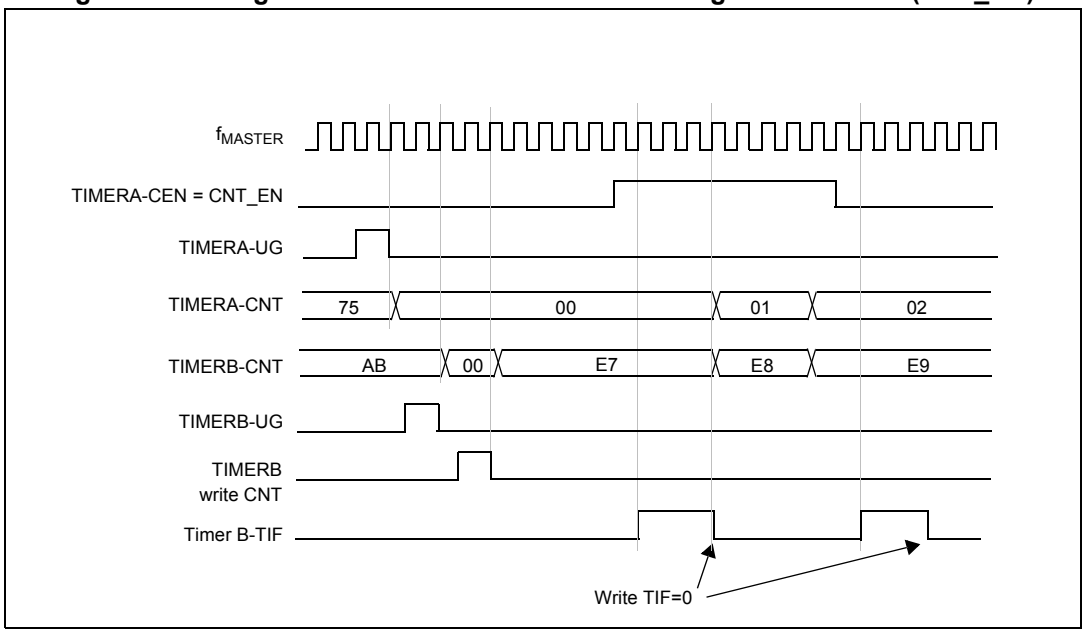


In the example in *Figure 43*, the Timer B counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer A. You can then write any value you want in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx_EGR registers.

In the next example, we synchronize Timer A and Timer B. Timer A is the master and starts from 0. Timer B is the slave and starts from E7h. The prescaler ratio is the same for both timers. Timer B stops when Timer A is disabled by writing ‘0’ to the CEN bit in the TIMx_CR1 register:

1. Configure Timer A master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIMx_CR2 register).
2. Configure the Timer A OC1REF waveform (TIMx_CCMR1 register).
3. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).
4. Configure Timer B in trigger gated mode (SMS=101 in TIMx_SMCR register).
5. Reset Timer A by writing ‘1’ in UG bit (TIMx_EGR register).
6. Reset Timer B by writing ‘1’ in UG bit (TIMx_EGR register).
7. Initialize Timer B to 0xE7 by writing ‘E7h’ in the Timer B counter (TIMx_CNTRL).
8. Enable Timer B by writing ‘1’ in the CEN bit (TIMx_CR1 register).
9. Start Timer A by writing ‘1’ in the CEN bit (TIMx_CR1 register).
10. Stop Timer A by writing ‘0’ in the CEN bit (TIMx_CR1 register).

Figure 44. Gating Timer B with the counter enable signal of Timer A (CNT_EN)

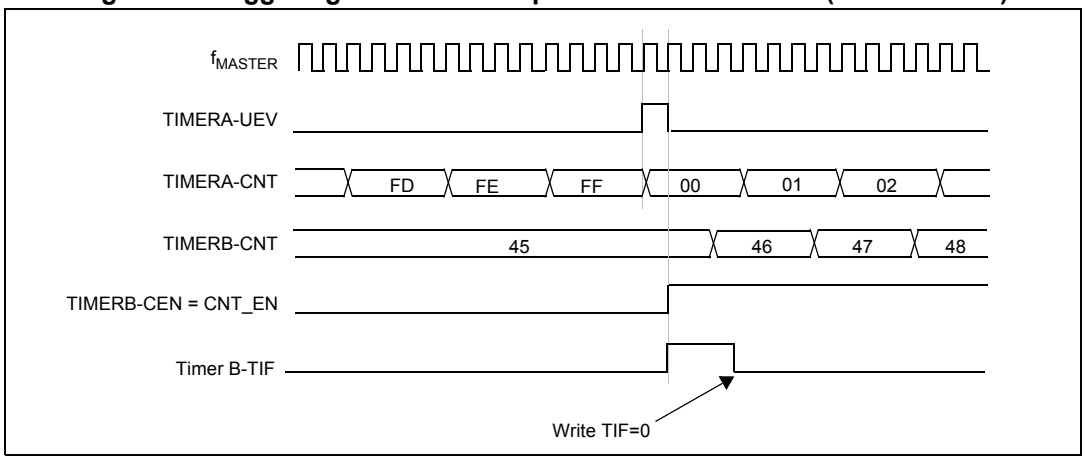


Using one timer to start another timer

In this example, we set the enable of Timer B with the update event of Timer A. Refer to *Figure 42* for connections. Timer B starts counting from its current value (which can be non-zero) on the divided internal clock as soon as the update event is generated by Timer A. When Timer B receives the trigger signal its CEN bit is automatically set and the counter counts until we write ‘0’ to the CEN bit in the TIMx_CR1 register. Both counter clock frequencies are divided by 4 by the prescaler compared to f_{MASTER} ($f_{CK_CNT} = f_{MASTER}/4$).

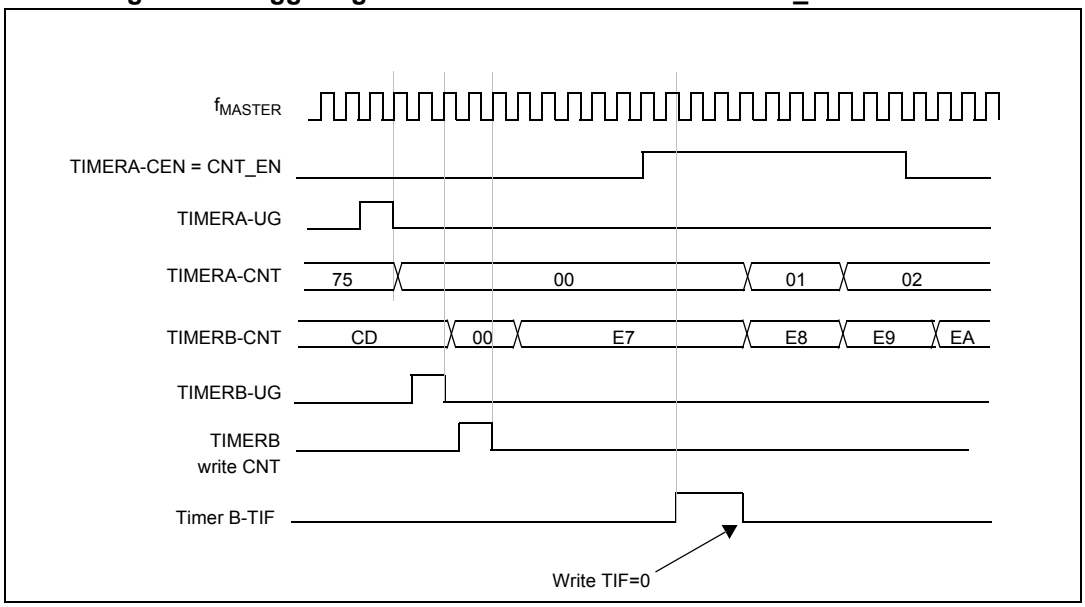
1. Configure Timer A master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIMx_CR2 register).
2. Configure the Timer A period (TIMx_ARR registers).
3. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).
4. Configure Timer B in trigger mode (SMS=110 in TIMx_SMCR register).
5. Start Timer A by writing ‘1’ in the CEN bit (TIMx_CR1 register).

Figure 45. Triggering Timer B with update event of Timer A (TIMERA-UEV)



As in the previous example, you can initialize both counters before starting counting. *Figure 46* shows the behaviour with the same configuration as in the *Figure 44* but in trigger standard mode instead of trigger gated mode (SMS=110 in the TIMx_SMCR register).

Figure 46. Triggering Timer B with counter enable CNT_EN of Timer A



Starting 2 timers synchronously in response to an external trigger

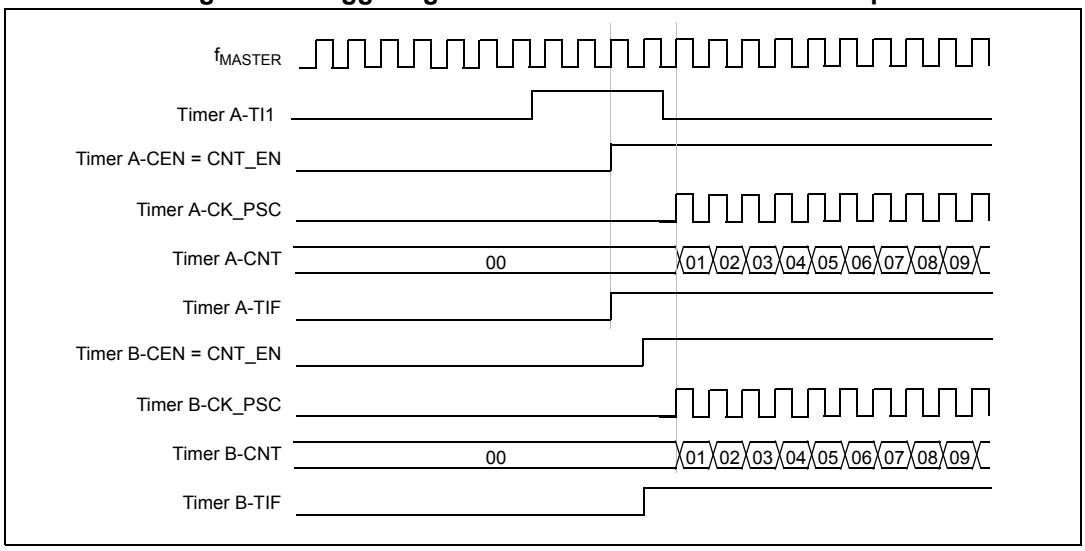
In this example, we set the enable of Timer A when its TI1 input rises, and the enable of Timer B with the enable of Timer A. Refer to *Figure 42* for connections. To ensure the counters alignment, Timer A must be configured in master/slave mode (slave with respect to TI1, master with respect to Timer B).

1. Configure Timer A master mode to send its Enable as trigger output (MMS=001 in the TIMx_CR2 register).
2. Configure Timer A slave mode to get the input trigger from TI1 (TS=100 in the TIMx_SMCR register).
3. Configure Timer A in trigger mode (SMS=110 in the TIMx_SMCR register).
4. Configure the Timer A in Master/Slave mode by writing MSM='1' (TIMx_SMCR register).
5. Configure Timer B to get the input trigger from Timer A (TS=001 in the TIMx_SMCR register).
6. Configure Timer B in trigger mode (SMS=110 in the TIMx_SMCR register).

When a rising edge occurs on TI1 (Timer A), both counters starts counting synchronously on the internal clock and both TIF flags are set.

Note: In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but you can easily insert an offset between them by writing any of the counter registers (TIMx_CNT). You can see that the master/slave mode insert a delay between CNT_EN and CK_PSC on Timer A.

Figure 47. Triggering Timer A and B with Timer A TI1 input

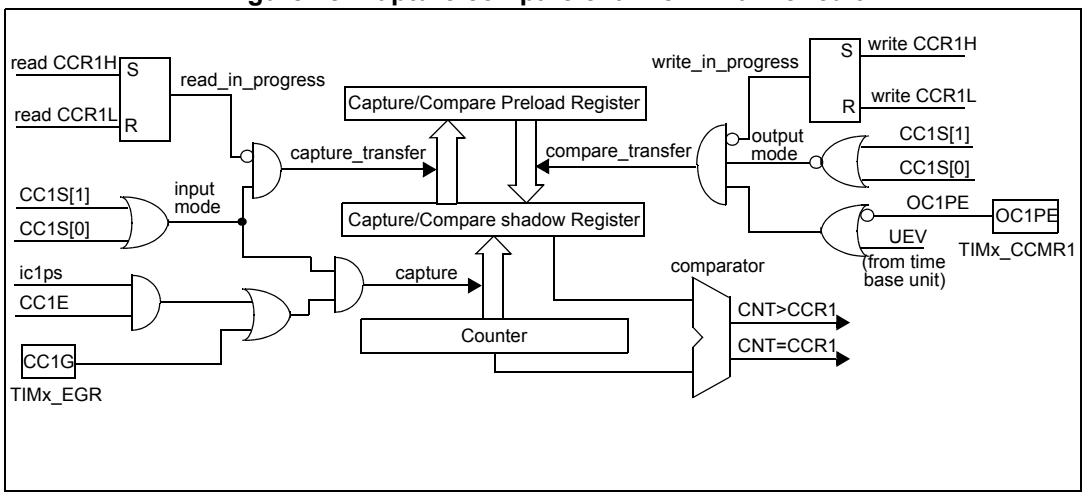


17.5 TIMx capture/compare channels

The timer I/O pins (TIMx_CH*i*) can be configured either for input capture or output compare functions. The choice made by configuring the CC*i*S channel selection bits in the capture/compare channel mode register (TIMx_CCMR*i*), where *i* is the channel number.

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 48. 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.

When the channel is configured in output mode (CC*i*S=0b00 in the TIMx_CCMR*i* register) where *i* is the channel number, the TIMx_CCR*i* register can be accessed without any restriction.

When the channel is configured in input mode, the sequence for reading the TIMx_CCR*i* register is the same as for the counter. See *Figure 49*. When a capture occurs, the content of the counter is captured into the TIMx_CCR*i* shadow register. Then this value is loaded into the preload register, except during the read sequence, when the preload register is frozen.

Figure 49. 16-bit read sequence for the TIMx_CCR*i* register in capture mode

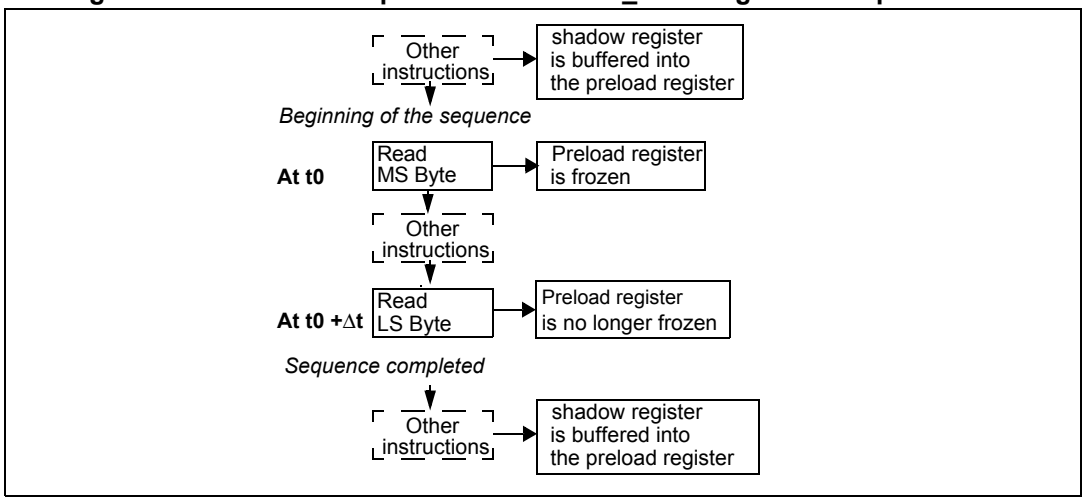


Figure 49 shows the sequence for reading the CCR*i* registers in the 16-bit timers. This buffered value remains unchanged until the 16-bit read sequence is completed.

After a complete reading sequence, if only the TIMx_CCR*i*L register is read, it returns the LS Byte of the count value at the time of the read.

If the MS byte is read after the LS byte, it no longer corresponds to the same captured value as the LS byte.

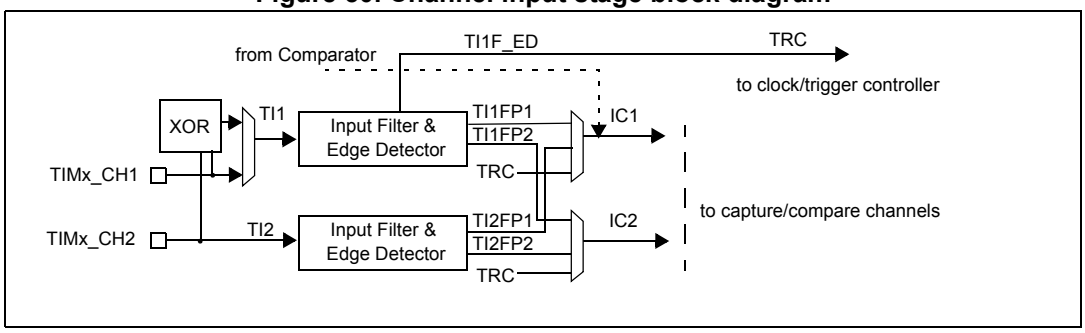
17.5.1 Write sequence for 16-bit TIMx_CCR*i* registers

16-bit values are loaded in the TIMx_CCR*i* registers through preload registers. This must be performed by two write instructions, one for each byte. The MS byte must be written first.

The shadow register update is blocked as soon as the MS byte has been written, and stays blocked until the LS byte has been written. Do not use the LDW instruction, as this writes the LS byte first, and would produce wrong results in this case.

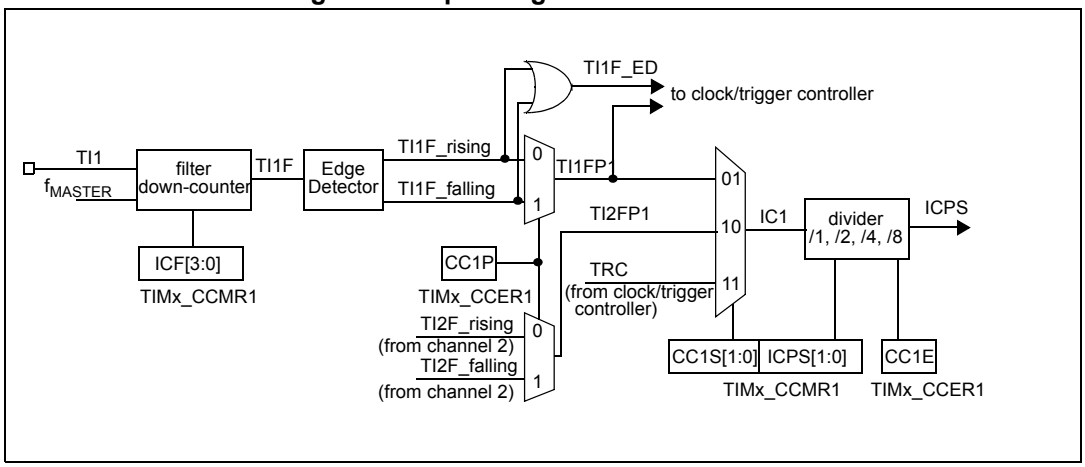
17.5.2 Input stage

Figure 50. Channel input stage block diagram



As shown in *Figure 51*, the input stage samples the corresponding TI*i* input to generate a filtered signal TI*i*F. Then, an edge detector with polarity selection generates a signal (TI*i*FPx) which can be used as trigger input by the clock/trigger controller or as the capture command. It is prescaled before the capture register (IC*i*PS).

Figure 51. Input stage of TIM 1 channel 1



17.5.3 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx_CCR*i*) are used to latch the value of the counter after a transition detected on the corresponding IC*i* signal. When a capture occurs, the corresponding CC*i*IF flag (TIMx_SR1 register) is set.

An interrupt can be sent if it is enabled by setting the CC*i*IE bit in the TIMx_IER register. If a capture occurs while the CC*i*IF flag was already high, then the over-capture flag CC*i*OF (TIMx_SR2 register) is set. CC*i*IF can be cleared by software by writing it to ‘0’ or by reading the captured data stored in the TIMx_CCR*i*L register. CC*i*OF is cleared when you write it to ‘0’.

The following example shows how to capture the counter value in TIMx_CCR1 when TI1 input rises. To do this, use the following procedure:

1. Select the active input: For example, to link the TIMx_CCR1 register to the TI1 input, write the CC1S bits to 0b01 in the TIMx_CCMR1 register. This configures the channel in input mode and the TIMx_CCR1 register becomes read-only.
2. Program the input filter duration that is needed for the type of the signal to be conntected to the timer. This is done for each TI*i* input using the IC*i*F bits in the TIMx_CCMR*i* register. For example, if you know that when, the input signal toggles, it is unstable for up to 5 f_{MASTER} cycles, you must program the filter duration longer than 5 clock cycles. The filter bits allow you to select a duration of 8 cycles by writing the value 0b0011 in in these bits the TIMx_CCMR1 register. With this filter setting, a transition on TI1 is valid only when 8 consecutive samples with the new level have been detected (sampled at f_{MASTER} frequency).
3. Select the edge of the active transition on the TI1 channel by writing CC1P bit to ‘0’ in the TIMx_CCER1 register (rising edge in this case).
4. Program the input prescaler. In our example, we want the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 0b00 in the TIMx_CCMR1 register).
5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER1 register.
6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_IER register.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- The input capture flag (CC1IF) is set (interrupt flag). The overcapture flag CC1OF is also set if at least two consecutive captures occured while the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.

To handle the overcapture event (CC1OF flag), 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.

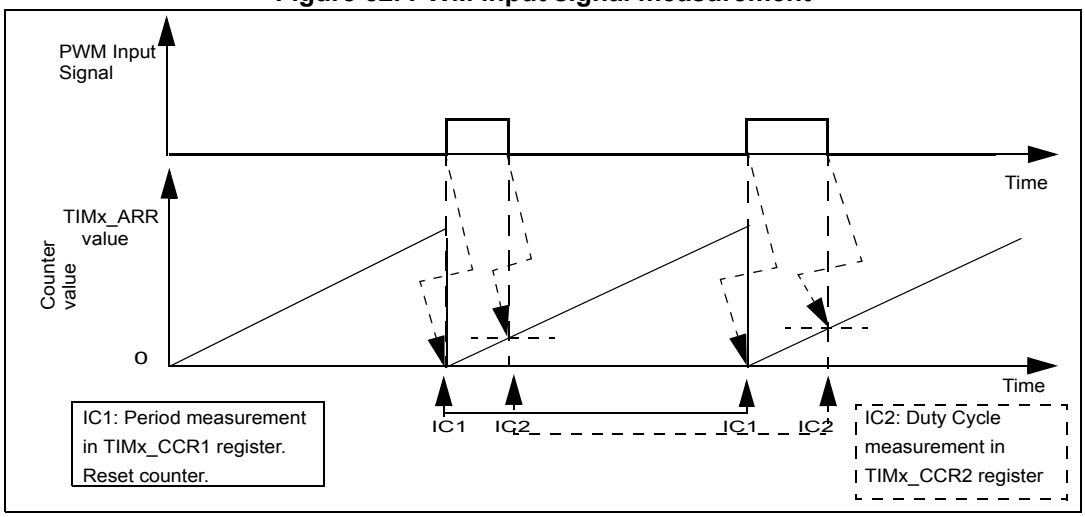
Note: It is not possible to send an IC interrupt without actually capturing the counter value in the TIMx_CCRx register. Nevertheless, it is possible to generate the capture event by software by setting the corresponding CCiG bit in the TIMx_EGR register.

PWM input signal measurement

This mode is a particular case of input capture mode. The procedure is the same except:

- Two IC*i* are mapped on the same TI*i* input.
- These 2 IC*i* are active on edges with opposite polarity.
- One of the two TI*i*FP is selected as trigger input and the clock/trigger controller is configured in trigger reset mode.

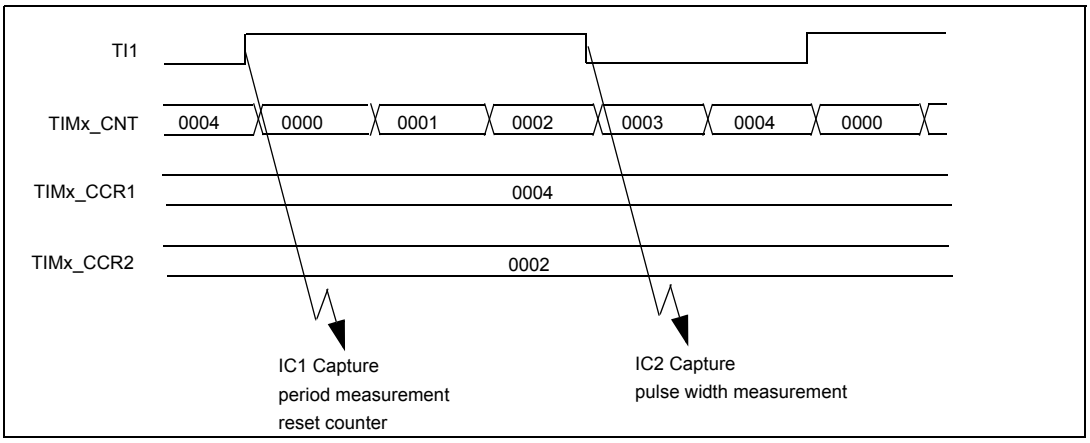
Figure 52. PWM input signal measurement



For example, you can measure the period (in the TIMx_CCR1 register) and the duty cycle (in the TIMx_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on f_{MASTER} frequency and prescaler value):

1. Select the active input capture or trigger input for TIMx_CCR1: write the CC1S bits to 0b01 in the TIMx_CCMR1 register (TI1FP1 selected).
2. 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).
3. Select the active input for TIMx_CCR2: write the CC2S bits to 0b10 in the TIMx_CCMR2 register (TI1FP2 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P bit to '1' (active on falling edge).
5. Select the valid trigger input: write the TS bits to 0b101 in the TIMx_SMCR register (TI1FP1 selected).
6. Configure the clock/trigger controller in reset mode: write the SMS bits to '100' in the TIMx_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER1 register.

Figure 53. PWM input signal measurement example



17.5.4 Output stage

The output stage generates an intermediate waveform called OC*i*REF (active high) which is then used for reference. Break functions and polarity act at the end of the chain.

Figure 54. Channel output stage block diagram

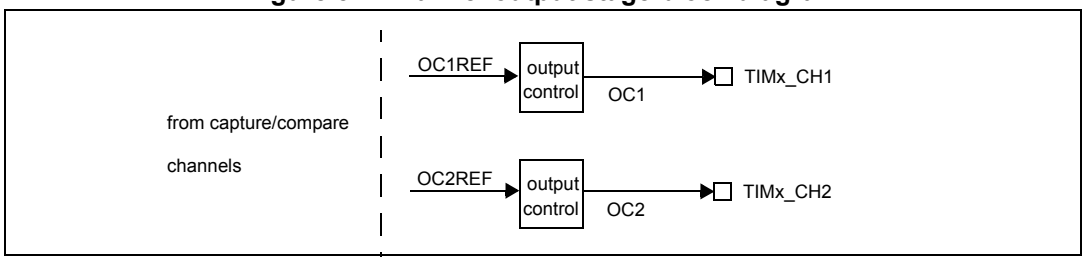
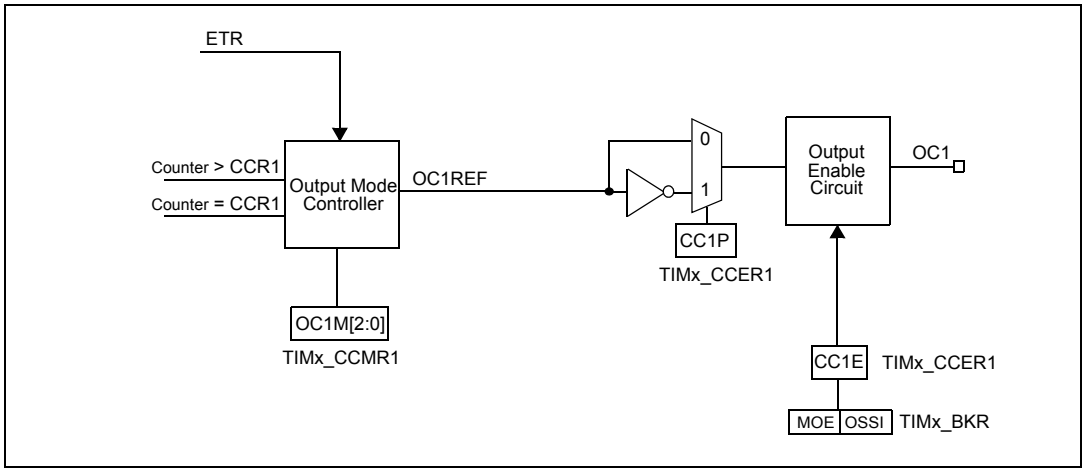


Figure 55. Output stage of channel 1



17.5.5 Forced output mode

In output mode (CC*i*S bits = 0b00 in the TIMx_CCMR*i* register) where *i* is the channel number, each output compare signal can be forced to high or low level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal to its active level, you just need to write '101' in the OC*i*M bits in the corresponding TIMx_CCMR*i* register. Thus OC*i*REF is forced high (OC*i*REF is always active high) and the OC*i* output is forced to high or low level depending on the CC*i*P polarity bit.

For example: CC*i*P=0 (OC*i* active high) => OC*i* is forced to high level.

The OC*i*REF signal can be forced low by writing the OC*i*M bits to 0b100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCR*i* 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.

17.5.6 Output compare mode

This function is used to control an output waveform or indicate when a period of time has elapsed.

When a match is found between the capture/compare register and the counter:

- Depending on the output compare mode, the corresponding OC*i* output pin:
  - keeps its level (OC*i*M=0b000),
  - is set active (OC*i*M=0b001),
  - is set inactive (OC*i*M=0b010)
  - or toggles (OC*i*M=0b011)
- Sets a flag in the interrupt status register (CC*i*IF bit in the TIMx_SR1 register).
- Generates an interrupt if the corresponding interrupt mask is set (CC*i*IE bit in the TIMx_IER register).

The output compare mode is defined by the OC*i*M bits in the TIMx_CCMR*i* register. The active or inactive level polarity is defined by the CC*i*P bit in the TIMx_CCER*i* register.

The TIMx_CCR*i* registers can be programmed with or without preload registers using the OC*i*PE bit in the TIMx_CCMR*i* register.

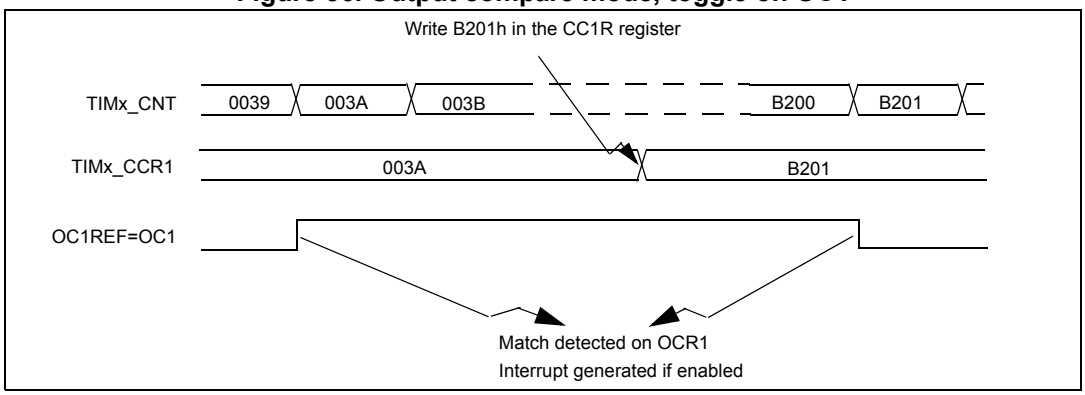
In output compare mode, the update event UEV has no effect on the OC*i*REF and OC*i* output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse.

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCR*i* registers.
3. Set the CC*i*IE bit if an interrupt request is to be generated.
4. Select the output mode as follwos:
   - Write OC*i*M = 0b011 to toggle OC*i* output pin when CNT matches CCR*i*
   - Write OC*i*PE = 0 to disable preload register
   - Write CC*i*P = 0 to select active high polarity
   - Write CC*i*E = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx_CR1 register.

The TIMx_CCR*i* register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OC*i*PE='0', else TIMx_CCR*i* shadow register will be updated only at the next update event UEV). An example is given in *Figure 56*.

Figure 56. Output compare mode, toggle on OC1



17.5.7 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCR*i* register.

The PWM mode can be selected independently on each channel (one PWM per OC*i* output) by writing 0b110 (PWM mode 1) or 0b111 (PWM mode 2) in the OC*i*M bits in the TIMx_CCMR*i* register. You must enable the corresponding preload register by setting the OC*i*PE bit in the TIMx_CCMR*i* register, and optionally enable the auto-reload preload register (in up-counting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OC*i* polarity is software programmable using the CC*i*P bit in the TIMx_CCER*i* register. It can be programmed as active high or active low. OC*i* output is enabled by a combination of CC*i*E, MOE, OIS*i*, OSSR and OSSI bits (TIMx_CCER*i* and TIMx_BKR registers). Refer to the TIMx_CCER*i* register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCR*i* are always compared to determine whether TIMx_CCR*i*≤TIMx_CNT or TIMx_CNT≤TIMx_CCR*i* (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.

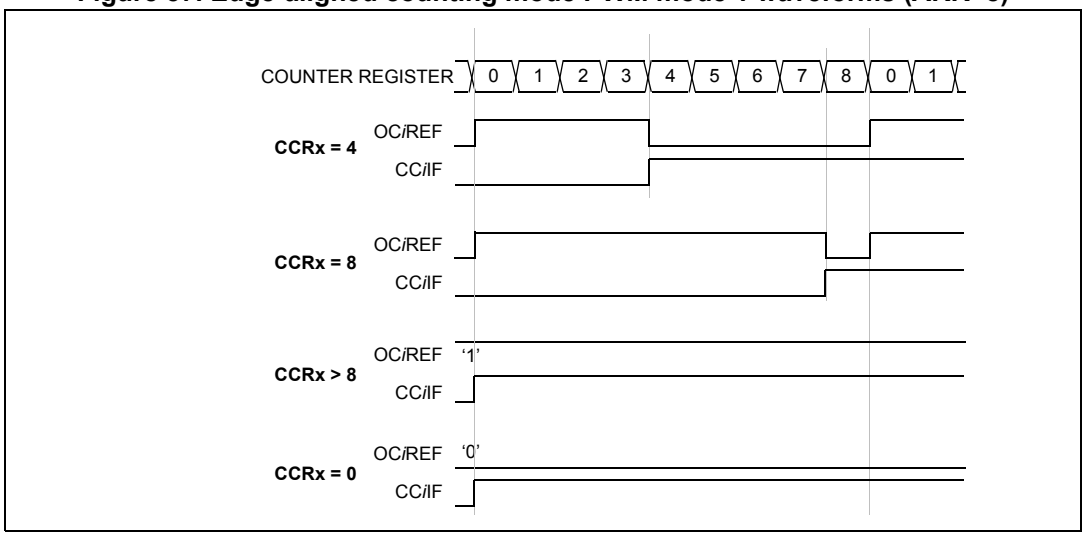
PWM edge-aligned mode

Up-counting configuration

Up-counting is active when the DIR bit in the TIMx_CR1 register is low.

In the following example, we consider the PWM mode 1. The reference PWM signal OC*i*REF is high as long as TIMx_CNT <TIMx_CCR*i* else it becomes low. If the compare value in TIMx_CCR*i* is greater than the auto-reload value (in TIMx_ARR) then OC*i*REF will be held at '1'. If the compare value is 0 then OC*i*REF will be held at '0'. *Figure 57* shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

Figure 57. Edge-aligned counting mode PWM mode 1 waveforms (ARR=8)



Down-counting configuration

Down-counting is active when DIR bit in TIMx_CR1 register is high. Refer to *Down-counting mode on page 111*

In PWM mode 1, the reference signal OC*i*REF is low as long as TIMx_CNT> TIMx_CCR*i* else it becomes high. If the compare value in TIMx_CCR*i* is greater than the auto-reload value in TIMx_ARR, then OC*i*REF will be held at '1'. 0% PWM is not possible in this mode.

PWM center-aligned mode

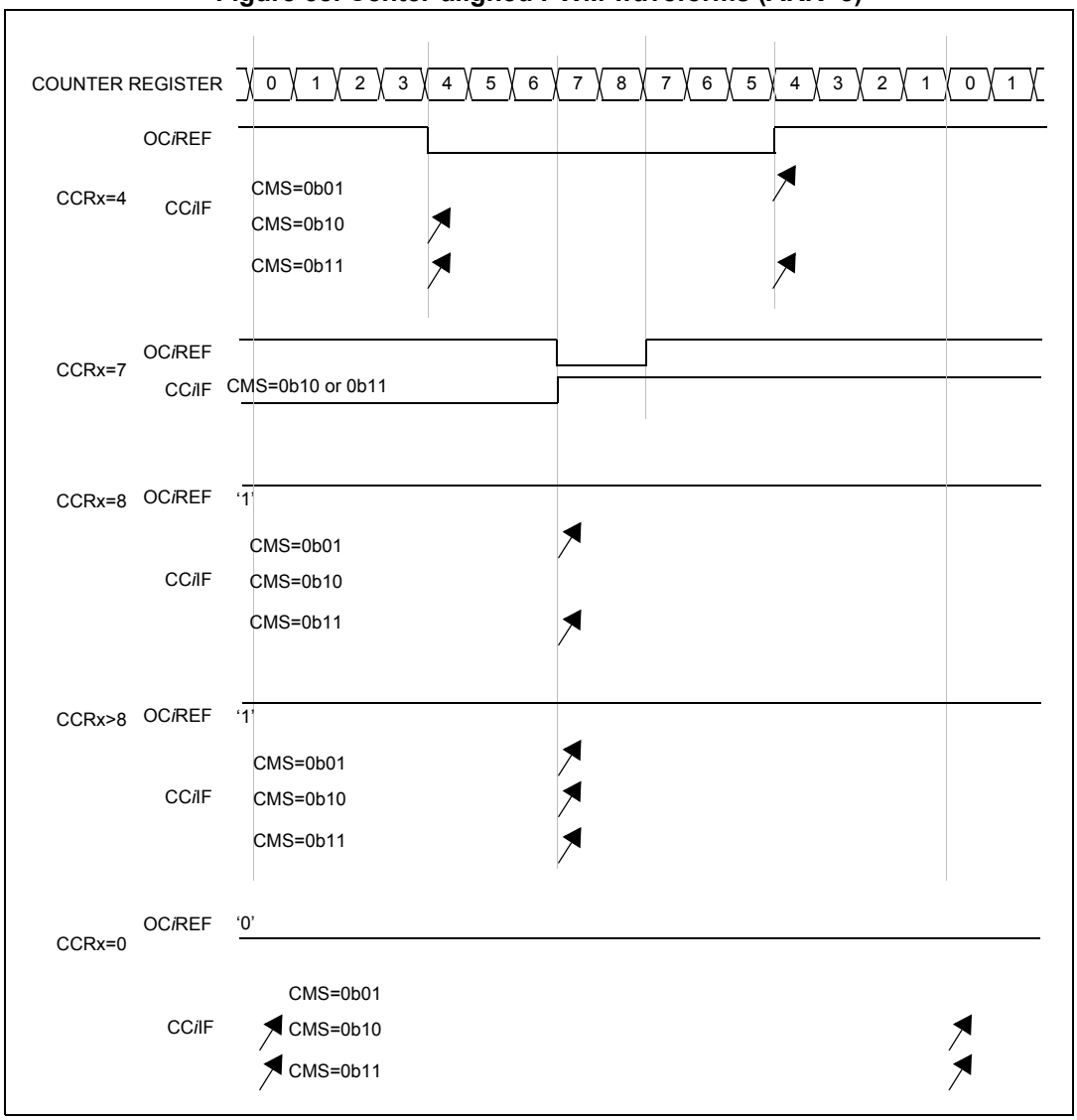
Center-aligned mode is active when the CMS bits in TIMx_CR1 register are different from '00' (all the remaining configurations having the same effect on the OC*i*REF/OC*i* signals).

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 is read-only in this mode. Refer to *Center-aligned mode (up/down counting) on page 113*.

Figure 58 shows some center-aligned PWM waveforms in an example where:

- The TIMx_ARR=8,
- PWM mode is PWM mode 1,
- the flag is set (arrow symbol in *Figure 58*) in three different cases:
  - only when the counter counts down (CMS=0b01)
  - only when the counter counts up (CMS=0b10) .
  - when the counter counts up and down (CMS=0b11) .

Figure 58. Center-aligned PWM waveforms (ARR=8)



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.

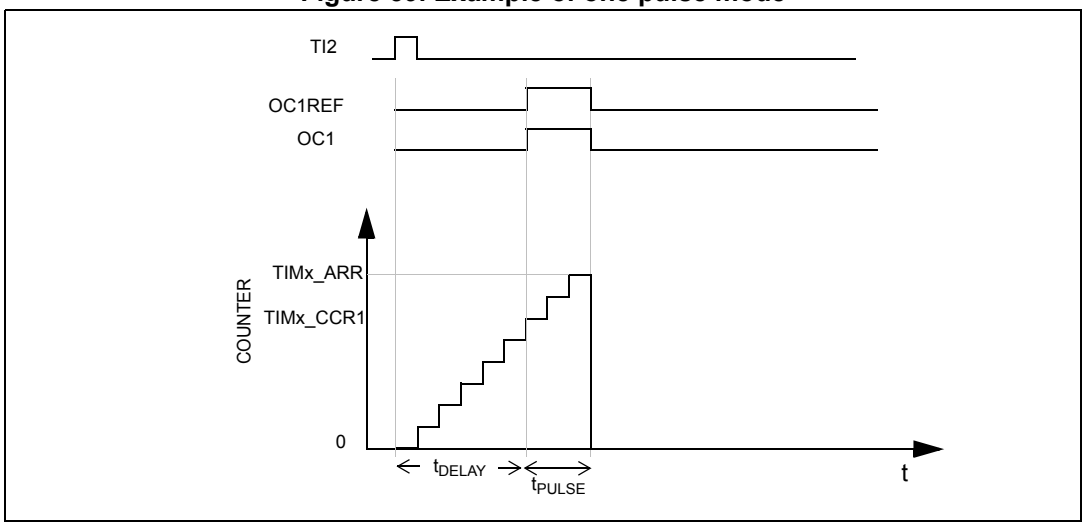
Starting the counter can be controlled through the clock/trigger controller. Generating the waveform can be done in output compare mode or PWM mode. You select One Pulse Mode by setting the OPM bit in the TIMx_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

In up-counting: CNT<CCR*i*≤ ARR (in particular, 0<CCR*i*),

In down-counting: CNT>CCR*i*.

Figure 59. Example of one pulse mode



For example you may want to generate a positive pulse on OC1 with a length of t_{PULSE} and after a delay of t_{DELAY} as soon as a positive edge is detected on the TI2 input pin.

Let's use IC2 as trigger 1:

- Map IC2 on TI2 by writing CC2S=0b01 in the TIMx_CCMR2 register.
- IC2 must detect a rising edge, write CC2P='0' in the TIMx_CCER1 register.
- Configure IC2 as trigger for the clock/trigger controller (TRGI) by writing TS=0b110 in the TIMx_SMCR register.
- IC2 is used to start the counter by writing SMS to 0b110 in the TIMx_SMCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler).

- The t_{DELAY} is defined by the value written in the TIMx_CCR1 register.
- The t_{PULSE} is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1).
- Let's say you want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this you enable PWM mode 2 by writing OC*i*M=0b111 in the TIMx_CCMR1 register. You can optionally enable the preload registers by writing OC1PE='1' in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case you have to write the compare value in the TIMx_CCR1 register, the auto-reload value in the TIMx_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx_CR1 register should be low.

You only want 1 pulse, so you write '1' in the OPM bit in the TIMx_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

Particular case: OC*i* fast enable:

In One Pulse Mode, the edge detection on TI*i* input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay t_{DELAY} min we can get.

If you want to output a waveform with the minimum delay, you can set the OC*i*FE bit in the TIMx_CCMR*i* register. Then OC*i*REF (and OC*i*) will be forced in response to the stimulus, without taking in account the comparison. Its new level will be the same as if a compare match had occured. OC*i*FE acts only if the channel is configured in PWM1 or PWM2 mode.

17.5.8 Using the break function

The break function is often used in motor control. When using the break function, the output enable signals and inactive levels are modified according to additional control bits (MOE, OSSR and OSSI bits in the TIMx_BKR register).

When exiting from reset, the break circuit is disabled and the MOE bit is low. You can enable the break function by setting the BKE bit in the TIMx_BKR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx_BKR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if you write MOE to 1 whereas it was low, you must insert a delay (dummy instruction) before reading it correctly.

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 OIS*i* bit in the TIMx_OISR register as soon as MOE=0. If OSSI=0 then the timer releases the enable output else the enable output remains high.
- The break status flag (BIF bit in the TIMx_SR1 register) is set. An interrupt can be generated if the BIE bit in the TIMx_IER register is set.
- If the AOE bit in the TIMx_BKR register is set, the MOE bit is automatically set again at the next update event UEV. This can be used to perform a regulation, for instance. Else, MOE remains low until you write it to '1' again. In this case, it can be used for security and you can connect the break input to an alarm from power drivers, thermal sensors or any security components.

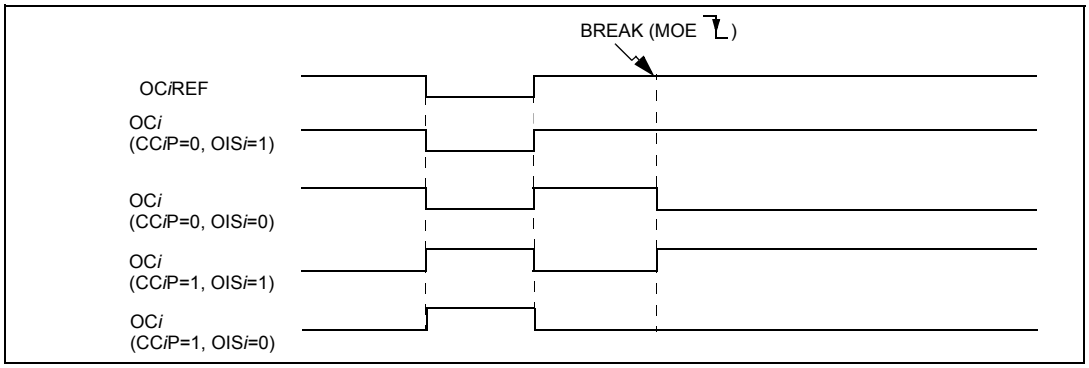
Note: The break inputs are acting on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.

The break can be generated by the break input (BKIN) which has a programmable polarity and can be enabled or disabled by setting or resetting the BKE bit in TIMx_BKR register.

In addition to the break inputs and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows you to freeze the configuration of several parameters (OC*i* polarities and state when disabled, OC*i*M configurations, break enable and polarity). You can choose from 3 levels of protection selected by the LOCK bits in the TIMx_BKR register. The LOCK bits can be written only once after an MCU reset.

Figure 60 shows an example of behavior of the outputs in response to a break.

Figure 60. Behavior of outputs in response to a break



17.5.9 Clearing the OC*i*REF signal on an external event

The OC*i*REF signal of a given channel can be cleared when a high level is detected on ETRF (if OC*i*CE='1' in the TIMx_CCMR*i* register, one enable bit per channel). The OC*i*REF signal remains low until the next UEV update event occurs. This function can be used in output compare mode and PWM mode only, it does not work in forced mode.

It can be connected to the output of a comparator and be used for current handling, for instance.

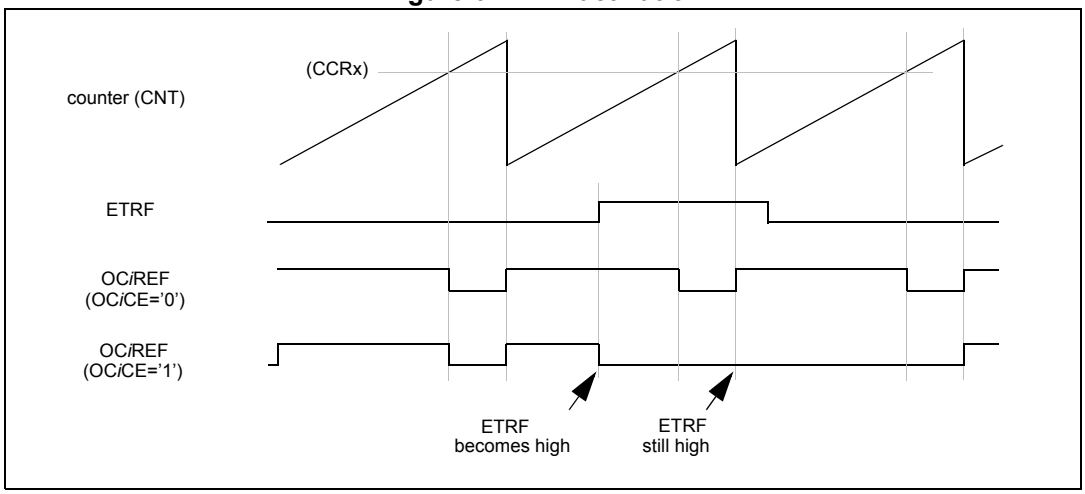
For example, the OC*i*REF signal can be connected to the output of a comparator to be used for current handling. In this case, the external trigger must be configured as follows:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] in the TIMx_ETR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE in the TIMx_ETR register set to '0'.
3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured as desired.

Refer to the external trigger input block diagram *Figure 34 on page 117*

Figure 61 shows the behavior of the OC*i*REF signal when the ETRF input becomes high, for both values of the enable bit OC*i*CE. In this example, the timer is programmed in PWM mode.

Figure 61. ETR activation



17.5.10 Encoder interface mode

This mode is typically used for motor control. To select Encoder Interface mode write SMS=0b001 in the TIMx_SMCR register if the counter is counting on TI2 edges only, SMS=0b010 if it is counting on TI1 edges only and SMS=0b011 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_CCER1 register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to *Table 34*. The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIMx_ARR before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

Table 34. Counting direction versus encoder signals

Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No count	No count
	Low	Up	Down	No count	No count
Counting on TI2 only	High	No count	No count	Up	Down
	Low	No count	No count	Down	Up
Counting on TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators will normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

The *Figure 62* 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 = 0b01 (TIMx_CCMR1 register, IC1 mapped on TI1).
- CC2S = 0b01 (TIMx_CCMR2 register, IC2 mapped on TI2).
- CC1P = 0 (TIMx_CCER1 register, IC1 non-inverted, IC1=TI1).
- CC2P = 0 (TIMx_CCER2 register, IC2 non-inverted, IC2=TI2).
- SMS = 0b011 (TIMx_SMCR register, both inputs are active on both rising and falling edges).
- CEN = 1 (TIMx_CR1 register, Counter is enabled).

Figure 62. Example of counter operation in encoder interface mode

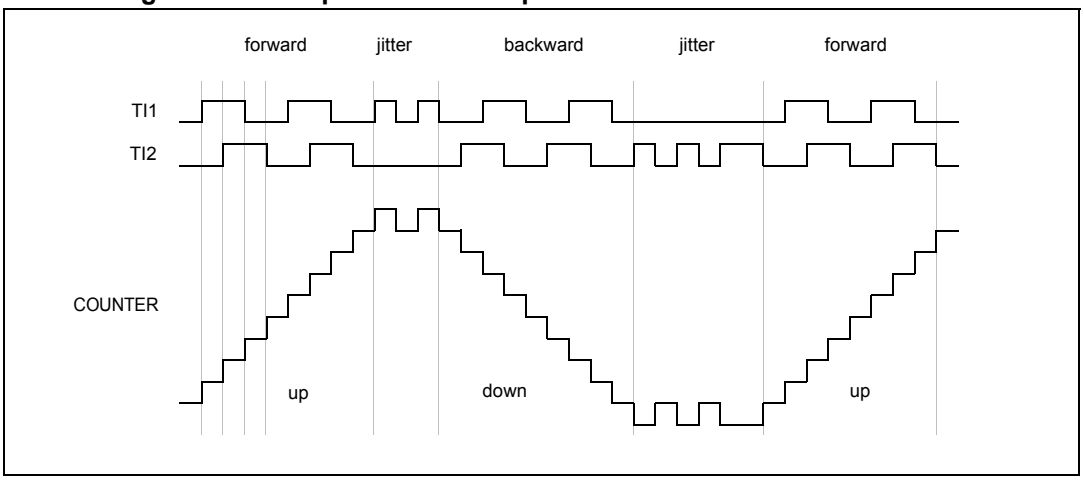
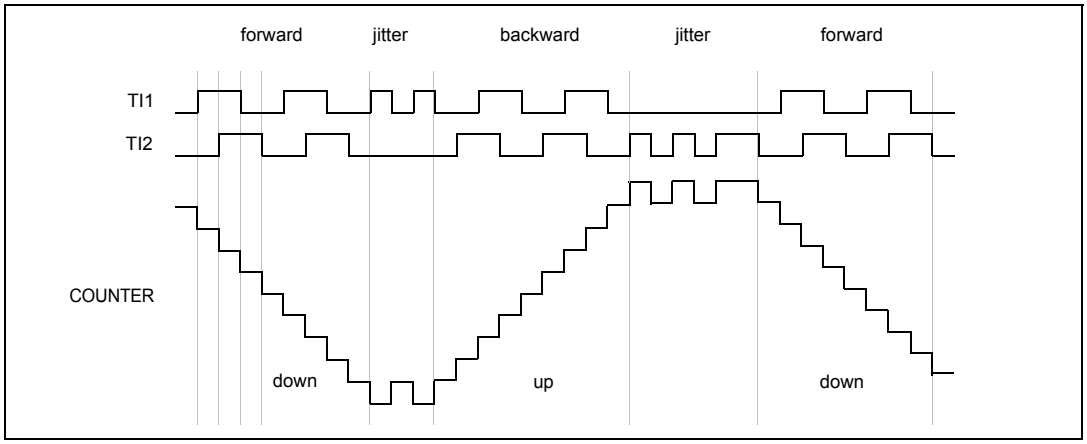


Figure 63 gives an example of counter behaviour when IC1 polarity is inverted (same configuration as above except CC1P='1').

Figure 63. Example of encoder interface mode with IC1 polarity inverted



The timer, when configured in Encoder Interface mode provides information on the sensor's current position. You can obtain dynamic information (speed, acceleration, decceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. You can do this by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer).

17.6 TIMx interrupts

TIMx has 6 interrupt request sources, mapped on 2 interrupt vectors:

- Break interrupt
- Trigger interrupt
- Commutation interrupt
- Capture/Compare 2 interrupt
- Capture/Compare 1 interrupt
- Update Interrupt (ex: overflow, underflow, counter initialization)

To use the interrupt features, for each interrupt channel used, set the desired “Interrupt Enable” bit: BIE, TIE, COMIE, CC*i*IE, UIE bits in the TIMx_IER register to enable interrupt requests.

The different interrupt sources can be also generated by software using the corresponding bits in the TIMx_EGR register.

17.6.1 TIMx wait-for-event capability

In wait-for-event mode (WFE), TIMx Capture/Compare, break, trigger and update interrupts can be used to wake up the device. The interrupt event must have been previously configured through bits TIMx_EV0 and TIMx_EV1 in the WFE_CR1 register (see *Section 9.5: WFE registers*).

17.7 TIMx registers

17.7.1 Control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
rw	rw		rw	rw	rw	rw	rw

Bit 7 **ARPE**: Auto-Reload Preload Enable

0: TIMx_ARR register is not buffered through a preload register. It can be written directly.

1: TIMx_ARR register is buffered through a preload register.

Bits 6:5 **CMS**: Center-aligned Mode Selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down 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)

Encoder mode (SMS=001, 010 or 011 in GPT_SMCR register) must be disabled in center-aligned mode.

Bit 4 **DIR**: Direction

0: Counter used as up-counter.

1: Counter used as down-counter.

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 CEN bit)

Bit 2 **URS**: Update Request Source

0: When enabled, an update interrupt request is sent as soon as registers are updated (counter overflow)

1: When enabled, an update interrupt request is sent only when the counter reaches the overflow.

Bit 1 **UDIS**: Update Disable

0: An Update event is generated as soon as a counter overflow occurs or a software update is generated or an hardware reset is generated by the clock/trigger mode controller. Buffered registers are then loaded with their preload values

1: An Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are re-initialized if the UG bit is set .

Bit 0 **CEN**: Counter Enable

0: Counter disabled

1: Counter enabled

17.7.2 Control register 2 (TIMx_CR2)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	MMS[2:0]			Reserved			
	rw						

Bit 7 Reserved

Bits 6:4 **MMS**: Master mode selection

These bits select the information to be sent in master mode to other timers for synchronization (TRGO). The combination is as follows:

000: **Reset** - the UG bit from the TIM3_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in trigger reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal is used as trigger output (TRGO). It is used 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 TIM3_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO).

011: Reserved

100: Reserved

101: Reserved

111: Reserved

Bits 3:0 Reserved

17.7.3 Slave mode control register (TIMx_SMCR)

Address offset: 0x02

Reset value: 0x00

7	6	5	4	3	2	1	0
MSM	TS[2:0]			Reserved	SMS[2:0]		
rw	rw				rw		

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 timers (through TRGO).

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 ITR0 connected to TIM4 TRGO
001: Reserved
010: Internal trigger ITR2 connected to TIM3 TRGO
011: Internal trigger ITR3 connected to TIM2 TRGO
100: TI1 Edge Detector (TI1F_ED)
101: Filtered Timer Input 1 (TI1FP1)
110: Filtered Timer Input 2 (TI2FP2)
111: External Trigger input (ETRF)

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

Bits 2:0 **SMS[2:0]**: Clock/trigger/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: Clock/trigger controler disabled - if CEN = ‘1’ then the prescaler is clocked directly by the internal clock.
001: Reserved
010: Reserved
011: Reserved
100: Trigger reset mode - Rising edge of the selected trigger signal (TRGI) reinitializes the counter and generates an update of the registers.
101: Gated Mode - The counter clock is enabled when the trigger signal (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.

17.7.4 External trigger register (TIMx_ETR)

Address offset: 0x03

Reset value: 0x00

7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]			
rw	rw	rw		rw			

Bit 7 **ETP**: External trigger polarity

This bit selects whether ETR or $\overline{ETR}$ is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 6 **ECE**: External clock enable

This bit enables External clock mode 2.

0: External clock mode 2 disabled.

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

Note: Setting the ECE bit has the same effect as selecting the external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111 in the TIMx_SMCR register).

It is possible to use simultaneously the external clock mode 2 with the following modes: trigger standard mode, trigger reset mode and trigger gated mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111 in TIMx_SMCR register).

If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input will be ETRF.

Bits 5:4 **ETPS**: External trigger prescaler

External trigger signal ETRP frequency must be at must 1/4 of f_{MASTER} 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 3:0 **ETF**: 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 events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{MASTER}

0001: $f_{SAMPLING}=f_{MASTER}$, N=2

0010: $f_{SAMPLING}=f_{MASTER}$, N=4

0011: $f_{SAMPLING}=f_{MASTER}$, N=8

0100: $f_{SAMPLING}=f_{MASTER}/2$, N=6

0101: $f_{SAMPLING}=f_{MASTER}/2$, N=8

0110: $f_{SAMPLING}=f_{MASTER}/4$, N=6

0111: $f_{SAMPLING}=f_{MASTER}/4$, N=8

1000: $f_{SAMPLING}=f_{MASTER}/8$, N=6

1001: $f_{SAMPLING}=f_{MASTER}/8$, N=8

1010: $f_{SAMPLING}=f_{MASTER}/16$, N=5

1011: $f_{SAMPLING}=f_{MASTER}/16$, N=6

1100: $f_{SAMPLING}=f_{MASTER}/16$, N=8

1101: $f_{SAMPLING}=f_{MASTER}/32$, N=5

1110: $f_{SAMPLING}=f_{MASTER}/32$, N=6

1111: $f_{SAMPLING}=f_{MASTER}/32$, N=8

17.7.5 Interrupt enable register (TIMx_IER)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
BIE	TIE	Reserved			CC2IE	CC1IE	UIE
rw	rw				rw	rw	rw

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.

Bits 5:3 Reserved

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

17.7.6 Status register 1 (TIMx_SR1)

Address offset: 0x05

Reset value: 0x00

7	6	5	4	3	2	1	0
BIF	TIF	Reserved			CC2IF	CC1IF	UIF
rc_w0	rc_w0				rc_w0	rc_w0	rc_w0

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 signal, both edges in case gated mode is selected). It is cleared by software.
0: No trigger event occurred.
1: Trigger interrupt pending.

Bits 5:3 Reserved

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. It is cleared by software.

0: No match.

1: The content of the counter TIMx_CNT has matched the content of the TIMx_CCR1 register.

– If channel CC1 is configured as input:

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1L 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 if UDIS=0 in the TIMx_CR1 register.

– When CNT is re-initialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

17.7.7 Status register 2 (TIMx_SR2)

Address offset: 0x06

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved					CC2OF	CC1OF	Reserved
					rc_w0	rc_w0	

Bits 7:3 Reserved

Bit 2 **CC2OF**: Capture/Compare 2 overcapture flag

Refer to CC1OF description

Bit 1 **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 0 Reserved

17.7.8 Event generation register (TIMx_EGR)

Address offset: 0x07

Reset value: 0x00

7	6	5	4	3	2	1	0
BG	TG	Reserved			CC2G	CC1G	UG
w	w				w	w	w

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. An interrupt is generated if enabled by the BIE bit.

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_SR1 register. An interrupt is generated if enabled by the TIE bit.

Bits 5:3 Reserved

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 the CC1 channel is configured in output mode**:
CC1IF flag is set, and the corresponding interrupt request is sent if enabled.
– **If the CC1 channel configured in input mode**:
The current value of the counter is captured in the TIMx_CCR1 register. The CC1IF flag is set, and the corresponding interrupt 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-initializes the counter and generates an update of the registers. Note that the prescaler counter is cleared too.

17.7.9 Capture/compare mode register 1 (TIMx_CCMR1)

The channel can be used in input (capture mode) or in output (compare mode). The direction of the channel is defined by configuring the CC1S 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 be aware that the same bit can have a different meaning for the input stage and for the output stage.

Address offset: 0x08
Reset value: 0x00

Channel configured in output

7	6	5	4	3	2	1	0
Reserved	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
	rw			rw	rw	rw	

Bit 7 Reserved

Bits 6:4 **OC1M**: Output compare 1 mode

These bits defines the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on the 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 matchs 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 matchs 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 up-counting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In down-counting, channel 1 is inactive (OC1REF=‘0’) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=’1’).

111 : PWM mode 2 - In up-counting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active.

Note: In PWM mode 1 or 2, the OCiREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode. Refer to Section 17.5.7 on page 134 for more details.

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 shadow register at each update event.

Note: For correct operation, preload registers must be enabled when the timer is in PWM mode. This is not mandatory in one pulse mode (OPM bit set in TIMx_CR1 register).

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 TI1FP1.

10: CC1 channel is configured as input, IC1 is mapped on TI2FP1.

11: Reserved

Note: CC1S bits are writable only when the channel is OFF (CC1E= '0' in TIMx_CCER1 and updated).

Channel configured in input

7	6	5	4	3	2	1	0
IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw				rw		rw	

Bits 7:4 **IC1F**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{MASTER}

0001: $f_{SAMPLING}=f_{MASTER}$, N=2

0010: $f_{SAMPLING}=f_{MASTER}$, N=4

0011: $f_{SAMPLING}=f_{MASTER}$, N=8

0100: $f_{SAMPLING}=f_{MASTER}/2$, N=6

0101: $f_{SAMPLING}=f_{MASTER}/2$, N=8

0110: $f_{SAMPLING}=f_{MASTER}/4$, N=6

0111: $f_{SAMPLING}=f_{MASTER}/4$, N=8

1000: $f_{SAMPLING}=f_{MASTER}/8$, N=6

1001: $f_{SAMPLING}=f_{MASTER}/8$, N=8

1010: $f_{SAMPLING}=f_{MASTER}/16$, N=5

1011: $f_{SAMPLING}=f_{MASTER}/16$, N=6

1100: $f_{SAMPLING}=f_{MASTER}/16$, N=8

1101: $f_{SAMPLING}=f_{MASTER}/32$, N=5

1110: $f_{SAMPLING}=f_{MASTER}/32$, N=6

1111: $f_{SAMPLING}=f_{MASTER}/32$, N=8

Bits 3:2 **IC1PSC**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).
The prescaler is reset as soon as CC1E=’0’ (TIMx_CCER register).
00: no prescaler, capture is done each time an edge is detected on the capture input.
01: Capture is done once every 2 events.
10: Capture is done once every 4 events.
11: Capture is done once every 8 events.

Note: The internal event counter is not reset when IC1PSC is changed on the fly. In this case the old value is used until the next capture occurs. To force a new value to be taken in account immediately, you can clear the CC1E bit and set it again.

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 TI1FP1.
10: CC1 channel is configured as input, IC1 is mapped on TI2FP1.
11: Reserved

Note: CC1S bits are writable only when the channel is OFF (CC1E= ’0’ in TIMx_CCER1 and updated).

17.7.10 Capture/compare mode register 2 (TIMx_CCMR2)

Note: Refer to Capture/compare mode register 1 (TIMx_CCMR1) for details on using these bits.

Address offset: 0x09
Reset value: 0x00

Channel configured in output

7	6	5	4	3	2	1	0
Reserved	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
	rw			rw	rw	rw	

Bit 7 Reserved

Bits 6:4 **OC2M**: Output compare 2 mode

Bit 3 **OC2PE**: Output compare 2 preload enable

Bit 2 **OC2FE**: Output compare 2 fast enable

Bits 1:0 **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 TI2FP2.
10: CC2 channel is configured as input, IC2 is mapped on TI1FP2.
11: Reserved

Note: CC2S bits are writable only when the channel is OFF (CC2E = ’0’ in TIMx_CCER1).

Channel configured in input

7	6	5	4	3	2	1	0
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	
rw				rw		rw	

Bits 7:4 **IC2F**: Input capture 2 filter

Bits 3:2 **IC2PSC**: Input capture 2 prescaler

Bits 1:0 **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 TI2FP2.
10: CC2 channel is configured as input, IC2 is mapped on TI1FP2.
11: Reserved
Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCER1).

17.7.11 Capture/compare enable register 1 (TIMx_CCER1)

Address offset: 0x0A

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
		rw	rw			rw	rw

Bits 6:7 Reserved

Bit 5 **CC2P**: Capture/Compare 2 output polarity
refer to CC1P description

Bit 4 **CC2E**: Capture/Compare 2 output enable
refer to CC1E description

Bits 2:3 Reserved

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 for capture function (see *Figure 51*):
0 : Capture is done on a rising edge of TI1F or TI2F
1 : Capture is done on a falling edge of TI1F or TI2F

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:
In this case 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.

17.7.12 Counter high (TIMx_CNTRH)

Address offset: 0x0B

Reset value: 0x00

7	6	5	4	3	2	1	0
CNT[15:8]							
rw							

Bits 7:0 **CNT[15:8]**: Counter value (MSB)

17.7.13 Counter low (TIMx_CNTRL)

Address offset: 0x0C

Reset value: 0x00

7	6	5	4	3	2	1	0
CNT[7:0]							
rw							

Bits 7:0 **CNT[7:0]**: Counter value (LSB)

17.7.14 Prescaler register (TIMx_PSCR)

Address offset: 0x0D

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved					PSC[2:0]		
					rw		

Bits 7:3 Reserved

Bits 2:0 **PSC[2:0]**: Prescaler value

The prescaler value divides the CK_PSC clock frequency.

The counter clock frequency f_{CK_CNT} is equal to $f_{CK_PSC} / 2^{(PSC[2:0])}$. PSC[7:3] are forced to 0 by hardware.

PSCR contains the value which will be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register).

This means that an update event must be generated in order that a new prescaler value can be taken into account.

17.7.15 Auto-reload register high (TIMx_ARRH)

Address offset: 0x0E

Reset value: 0xFF

7	6	5	4	3	2	1	0
ARR[15:8]							
rw							

Bits 7:0 **ARR[15:8]**: Autoreload value (MSB)

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 17.3: TIMx time base unit on page 107* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is 0.

17.7.16 Auto-reload register low (TIMx_ARRL)

Address offset: 0x0F

Reset value: 0xFF

7	6	5	4	3	2	1	0
ARR[7:0]							
rw							

Bits 7:0 **ARR[7:0]**: Autoreload value (LSB)

17.7.17 Capture/compare register 1 high (TIMx_CCR1H)

Address offset: 0x10

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR1[15:8]							
rw							

Bits 7:0 **CCR1[15:8]**: Capture/Compare 1 value (MSB)

If the CC1 channel is configured as output (CC1S bits in TIMx_CCMR1 register):
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 signalled on OC1 output.

If the CC1 channel is configured as input (CC1S bits in TIMx_CCMR1 register):
CCR1 is the counter value transferred by the last input capture 1 event (IC1). It is read-only in this case.

17.7.18 Capture/compare register 1 low (TIMx_CCR1L)

Address offset: 0x11
Reset value: 0x00

7	6	5	4	3	2	1	0
CCR1[7:0]							
rw							

Bits 7:0 **CCR1[7:0]**: Capture/Compare 1 value (LSB)

17.7.19 Capture/compare register 2 high (TIMx_CCR2H)

Address offset: 0x12

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR2[15:8]							
rw							

Bits 7:0 **CCR2[15:8]**: Capture/Compare 2 value (MSB)

If the CC2 channel is configured as output (CC2S bits in TIMx_CCMR2 register):
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 the CC2 channel is configured as input (CC2S bits in TIMx_CCMR2 register):
CCR2 is the counter value transferred by the last input capture 2 event (IC2).

17.7.20 Capture/compare register 2 low (TIMx_CCR2L)

Address offset: 0x13

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR2[7:0]							
rw							

Bits 7:0 **CCR2[7:0]**: Capture/Compare value (LSB)

17.7.21 Break register (TIMx_BKR)

Address offset: 0x14

Reset value: 0x00

7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	Reserved	OSSI	LOCK	
rw	rw	rw	rw		rw	rw	

Bit 7 **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 outputs are disabled or forced to idle state.

1: OC outputs are enabled if their respective enable bits are set (CCxE in TIMx_CCERx registers).

See OC enable description for more details (*Table 35: Output control bit for OCx channels with break feature on page 160*).

Bit 6 **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 no longer be modified as long as LOCK level 1 has been programmed (LOCK bits in the TIMx_BKR register).

Bit 5 **BKP**: Break polarity

0: Break input BKIN is active low

1: Break input BKIN is active high

Note: This bit can no longer be modified as long as LOCK level 1 has been programmed (LOCK bits in the TIMx_BKR register).

Bit 4 **BKE**: Break enable

0: Break input (BKIN) disabled

1: Break input (BKIN) enabled

Note: This bit can no longer be modified as long as LOCK level 1 has been programmed (LOCK bits in the TIMx_BKR register).

Bit 3 Reserved

Bit 2 **OSSI**: Off-State selection for idle mode

This bit is used when MOE=0 on channels configured as outputs.

See OC enable description for more details (*Table 35: Output control bit for OCx channels with break feature on page 160*).

0: When inactive, OCx outputs are disabled (OCx enable output signal=0).

1: When inactive, OCx outputs are forced first with their idle level as soon as CCxE=1. OC enable output signal=1).

Note: This bit can no longer be modified as soon as the LOCK level 2 has been programmed (LOCK bits in the TIMx_BKR register).

Bits 1:0 **LOCK**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bits are write protected.

01: LOCK Level 1 = OISx bit in TIMx_OISR register and BKE/BKP/AOE bits in TIMx_BKR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP bits in TIMx_CCERx registers, as long as the related channel is configured in output through the CCxS bits) as well as the 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_BKR register has been written, their content is frozen until the next reset.

Note: As the bits AOE, BKP, BKE and OSSI 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_BKR register.

Table 35. Output control bit for OCx channels with break feature

Control bits			OCx/OCx_EN output state
MOE bit	OSSI bit	CCxE bit	
1	X	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0
		1	OCx= OCxREF + Polarity (OCxREF xor CCxP) OCx_EN=1
0	0	0	Output Disabled (not driven by the timer) OCx=OISx, OCx_EN=0
	0	1	
	1	0	
	1	1	Off-State (output enabled with inactive state) OCx=OISx, OCx_EN=1

Note: The state of the external I/O pins connected to the OCx channels depends on the OCx channel state and the GPIO registers.

17.7.22 Output idle state register (TIMx_OISR)

Address offset: 0x15

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved					OIS2	Reserved	OIS1
					rw		rw

Bits 7:3 Reserved

Bit 2 **OIS2**: Output idle state 2 (OC2 output).
Refer to OIS1 bit

Bit 1 Reserved

Bit 0 **OIS1**: Output idle state 1 (OC1 output).
0: OC1=0 when MOE=0
1: OC1=1 when MOE=0

Note: This bit can no longer be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in the TIMx_BKR register).

17.7.23 TIMx register map and reset values

Refer to the datasheet for the base addresses of TIM2 and TIM3.

Table 36. TIMx register map

Address offset	Register Name	7	6	5	4	3	2	1	0
0x00	TIMx_CR1 Reset value	ARPE 0	CMS1 0	CMS0 0	DIR 0	OPM 0	URS 0	UDIS 0	CEN 0
0x01	TIMx_CR2 Reset value	TI1S 0	MMS2 0	MMS1 0	MMS0 0	- 0	- 0	- 0	- 0
0x02	TIMx_SMCR Reset value	MSM 0	TS2 0	TS1 0	TS0 0	- 0	SMS2 0	SMS1 0	SMS0 0
0x03	TIMx_ETR Reset value	ETP 0	ECE 0	ETPS1 0	ETPS0 0	EFT3 0	EFT2 0	EFT1 0	EFT0 0
0x04	TIMx_IER Reset value	BIE 0	TIE 0	- 0	- 0	- 0	CC2IE 0	CC1IE 0	UIE 0
0x05	TIMx_SR1 Reset value	BIF 0	TIF 0	- 0	- 0	- 0	CC2IF 0	CC1IF 0	UIF 0
0x06	TIMx_SR2 Reset value	- 0	- 0	- 0	- 0	- 0	CC2OF 0	CC1OF 0	- 0
0x07	TIMx_EGR Reset value	BG 0	TG 0	- 0	- 0	- 0	CC2G 0	CC1G 0	UG 0

Table 36. TIMx register map (continued)

Address offset	Register Name	7	6	5	4	3	2	1	0
0x08	TIMx_CCMR1 (output mode)	-	OC1M2	OC1M1	OC1M0	OC1PE	OC1FE	CC1S1	CC1S0
	Reset Value	0	0	0	0	0	0	0	0
	TIMx_CCMR1 (input mode)	IC1F3	IC1F2	IC1F1	IC1F0	IC1PSC1	IC1PSC0	CC1S1	CC1S0
	Reset value	0	0	0	0	0	0	0	0
0x09	TIMx_ CCMR2 (output mode)	-	OC2M2	OC2M1	OC2M0	OC2PE	OC2FE	CC2S1	CC2S0
	Reset value	0	0	0	0	0	0	0	0
	TIMx_CCMR2 (input mode)	IC2F3	IC2F2	IC2F1	IC2F0	IC2PSC1	IC2PSC0	CC2S1	CC2S0
	Reset value	0	0	0	0	0	0	0	0
0x0A	TIMx_CCER1	-	-	CC2P	CC2E	-	-	CC1P	CC1E
	Reset value	0	0	0	0	0	0	0	0
0x0B	TIMx_CNTRH	CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8
	Reset value	0	0	0	0	0	0	0	0
0x0C	TIMx_CNTRL	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
	Reset value	0	0	0	0	0	0	0	0
0x0D	TIMx_PSCR	-	-	-	-	-	PSC2	PSC1	PSC0
	Reset value	0	0	0	0	0	0	0	0
0x0E	TIMx_ARRH	ARR15	ARR14	ARR13	ARR12	ARR11	ARR10	ARR9	ARR8
	Reset value	1	1	1	1	1	1	1	1
0x0F	TIMx_ARRL	ARR7	ARR6	ARR5	ARR4	ARR3	ARR2	ARR1	ARR0
	Reset value	1	1	1	1	1	1	1	1
0x10	TIMx_CCR1H	CCR115	CCR114	CCR113	CCR112	CCR111	CCR110	CCR19	CCR18
	Reset value	0	0	0	0	0	0	0	0
0x11	TIMx_CCR1L	CCR17	CCR16	CCR15	CCR14	CCR13	CCR12	CCR11	CCR10
	Reset value	0	0	0	0	0	0	0	0
0x12	TIMx_CCR2H	CCR215	CCR214	CCR213	CCR212	CCR211	CCR210	CCR29	CCR28
	Reset value	0	0	0	0	0	0	0	0
0x13	TIMx_CCR2L	CCR27	CCR26	CCR25	CCR24	CCR23	CCR22	CCR21	CCR20
	Reset value	0	0	0	0	0	0	0	0
0x14	TIMx_BKR	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK	LOCK
	Reset value	0	0	0	0	0	0	0	0
0x15	TIMx_OISR	-	-	-	-	-	OIS2	-	OIS1
	Reset value	0	0	0	0	0	0	0	0

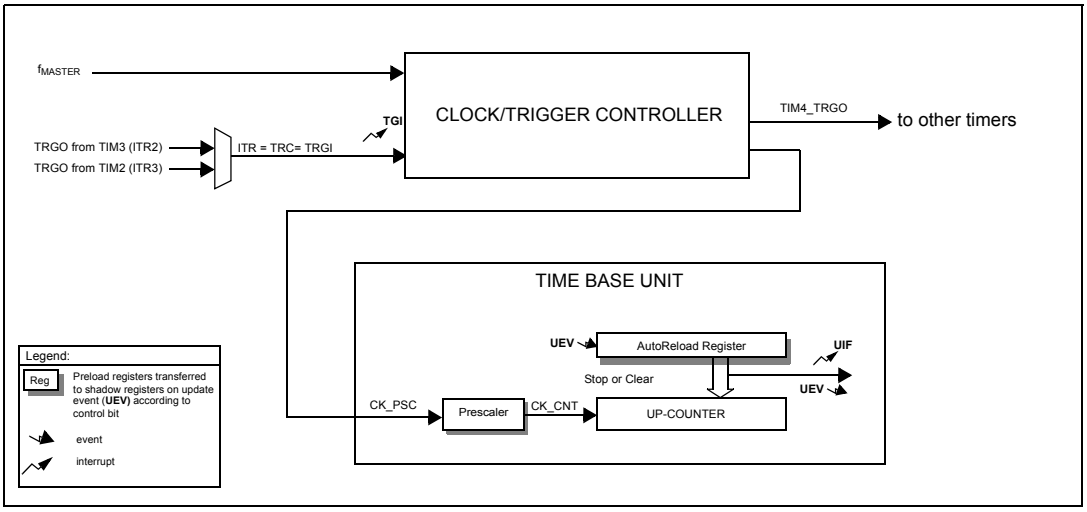
18 8-bit basic timer (TIM4)

18.1 Introduction

The timer consists of an 8-bit auto-reload up-counter driven by a programmable prescaler. It can be used for time base generation, with interrupt generation on timer overflow.

Refer to *Section 17.3 on page 107* for the general description of the timer features.

Figure 64. TIM4 block diagram



18.2 TIM4 main features

The main features include:

- 8-bit up counter auto-reload counter
- 4-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency by any power of 2 from 1 to 32768.
- Interrupt generation
  - On counter update: counter overflow
  - On trigger input

18.3 TIM4 interrupts

The timer has 2 interrupt request sources:

- Update Interrupt (overflow, counter initialization)
- Trigger input

18.4 TIM4 clock selection

The clock source for the timer is the internal clock (f_{MASTER}). It is connected directly to the CK_PSC clock that feeds the prescaler driving the counter clock CK_CNT.

Prescaler

The prescaler implementation is as follows:

- The TIM4 prescaler is based on a 16-bit counter controlled through a 4-bit register (in TIM4_PSCR register). It can be changed on the fly as this control register is buffered. It can divide the counter clock frequency by any power of 2 from 1 to 32768.

  The counter clock frequency is calculated as follows:

  $f_{CK_CNT} = f_{CK_PSC}/2^{(PSCR[3:0])}$

The prescaler value is loaded through a preload register. The shadow register, which contains the current value to be used is loaded as soon as the LS Byte has been written.

Read operations to the TIM4_PSCR registers access the preload registers, so no special care needs to be taken to read them.

18.5 TIM4 registers

18.5.1 Control register 1 (TIM4_CR1)

Address offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
ARPE	Reserved			OPM	URS	UDIS	CEN
rw				rw	rw	rw	rw

Bit 7 **ARPE**: Auto-Reload Preload Enable.

0: TIM4_ARR register is not buffered through a preload register. It can be written directly.

1: TIM4_ARR register is buffered through a preload register.

Bits 6:4 Reserved

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.

0: When enabled, an update interrupt request is sent as soon as registers are updated (counter overflow)

1: When enabled, an update interrupt request is sent only when the counter reaches the overflow/underflow.

Bit 1 **UDIS**: Update disable.

0: An Update event is generated as soon as a counter overflow occurs or a software update is generated. Buffered registers are then loaded with their preload values

1: An Update event is not generated, shadow registers keep their value (ARR, PSC). The counter and the prescaler are re-initialized if the UG bit is set.

Bit 0 **CEN**: Counter enable.

0: Counter disable.

1: Counter enable.

18.5.2 Control register 2 (TIM4_CR2)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	MMS[2:0]			Reserved			
	rw						

Bit 7 Reserved

Bits 6:4 **MMS**: Master mode selection

These bits select the information to be sent in master mode to other timers for synchronization (TRGO). The combination is as follows:

000: **Reset** - the UG bit in the TIM4_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in trigger reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal is used as trigger output (TRGO). It is used to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in the TIM4_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO).

011: Reserved

100: Reserved

101: Reserved

110: Reserved

111: Reserved

Bits 3:0 Reserved

18.5.3 Slave mode control register (TIM4_SMCR)

Address offset: 0x02

Reset value: 0x00

7	6	5	4	3	2	1	0
MSM	TS[2:0]			Reserved	SMS[2:0]		
rw	rw				rw		

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 timers (through TRGO).

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: Reserved

010: Internal trigger ITR2 connected to TIM3 TRGO

011: Internal trigger ITR3 connected to TIM2 TRGO

100: Reserved

101: Reserved

110: Reserved

111: Reserved

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

Bits 2:0 **SMS[2:0]**: Clock/trigger/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: Clock/trigger controller disabled - if CEN = ‘1’ then the prescaler is clocked directly by the internal clock.

001: Reserved

010: Reserved

011: Reserved

100: Trigger reset mode - Rising edge of the selected trigger signal (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger signal (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.

18.5.4 Interrupt enable register (TIM4_IER)

Address offset: 0x03

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TIE	Reserved					UIE
	rw						rw

Bit 7 Reserved

Bit 6 **TIE**: Trigger interrupt enable
0: Trigger interrupt disabled
1: Trigger interrupt enabled

Bits 5:1 Reserved

Bit 0 **UIE**: Update Interrupt Enable
0: Update Interrupt disabled
1: Update Interrupt enabled

18.5.5 Status register 1 (TIM4_SR1)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TIF	Reserved					UIF
	rc_w0						rc_w0

Bit 7 Reserved

Bit 6 **TIF**: Trigger interrupt flag
This flag is set by hardware on trigger event (active edge detected on TRGI signal, both edges in case gated mode is selected). It is cleared by software.
0: No trigger event occurred.
1: Trigger interrupt pending.

Bits 5:1 Reserved

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 if UDIS=0 in the TIM4_CR1 register.
- when CNT is re-initialized by software using the UG bit in the TIM4_EGR register, if URS=0 and UDIS=0 in the TIM4_CR1 register.

18.5.6 Event generation register (TIM4_EGR)

Address offset: 0x05

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TG	Reserved					UG
	w						w

Bit 7 Reserved

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 TIM4_SR1 register. An interrupt is generated if enabled by the TIE bit.

Bits 5:1 Reserved

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 counter and generates an update of the registers. Note that the prescaler counter is cleared too.

18.5.7 Counter (TIM4_CNTR)

Address offset: 0x06

Reset value: 0x00

7	6	5	4	3	2	1	0
CNT[7:0]							
rw							

Bits 7:0 CNT[7:0]: Counter value

18.5.8 Prescaler register (TIM4_PSCR)

Address offset: 0x07

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				PSC[3:0]			
				rw			

Bits 7:4 Reserved

Bits 3:0 **PSC[3:0]**: Prescaler value
The prescaler value divides the CK_PSC clock frequency.
The counter clock frequency f_{CK_CNT} is equal to $f_{CK_PSC} / 2^{(PSC[3:0])}$.
PSC contains the value which will be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIM4_EGR register).
This means that an update event must be generated in order that a new prescaler value can be taken into account.

18.5.9 Auto-reload register (TIM4_ARR)

Address offset: 0x08

Reset value: 0xFF

7	6	5	4	3	2	1	0
ARR[7:0]							
rw							

Bits 7:0 ARR[7:0]: Autoreload Value

18.5.10 TIM4 register map and reset values

Refer to the datasheet for the base address.

Table 37. TIM4 register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	TIM4_CR1 Reset value	ARPE 0	- 0	- 0	- 0	OPM 0	URS 0	UDIS 0	CEN 0
0x01	TIM4_CR2 Reset value	- 0	MMS2 0	MMS1 0	MMS0 0	- 0	- 0	- 0	- 0
0x02	TIM4_SMCR Reset value	MSM 0	TS2 0	TS1 0	TS0 0	- 0	SMS2 0	SMS1 0	SMS0 0
0x03	TIM4_IER Reset value	- 0	TIE 0	- 0	- 0	- 0	- 0	- 0	UIE 0
0x04	TIM4_SR1 Reset value	- 0	TIF 0	- 0	- 0	- 0	- 0	- 0	UIF 0
0x05	TIM4_EGR Reset value	- 0	TG 0	- 0	- 0	- 0	- 0	- 0	UG 0
0x06	TIM4_CNTR Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x07	TIM4_PSCR Reset value	- 0	- 0	- 0	- 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0
0x08	TIM4_ARR Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1

19 Inter-integrated circuit (I^2C) interface

19.1 Introduction

I^2C (inter-integrated circuit) bus interface serves as an interface between the microcontroller and the serial I^2C bus. It provides multi-master capability, and controls all I^2C bus-specific sequencing, protocol, arbitration and timing. It supports standard and fast speed modes.

19.2 I^2C main features

- Parallel-bus/I^2C protocol converter
- Multi-master capability: the same interface can act as Master or Slave
- I^2C Master features:
  - Clock generation
  - Start and Stop generation
- I^2C Slave features:
  - Programmable I^2C Address detection
  - Stop bit detection
  - I²C dual addressing capability to acknowledge 2 slave addresses
- Generation and detection of 7-bit/10-bit addressing and general call
- Supports different communication speeds:
  - Standard speed (up to 100 kHz),
  - Fast speed (up to 400 kHz)
- Status flags:
  - Transmitter/receiver mode flag
  - End-of-byte transmission flag
  - I^2C busy flag
- Error flags:
  - Arbitration lost condition for master mode
  - Acknowledgement failure after address/ data transmission
  - Detection of misplaced start or stop condition
  - Overrun/underrun if clock stretching is disabled
- 3 types of interrupts:
  - 1 communication interrupt
  - 1 error condition interrupt
  - 1 wakeup from Halt interrupt
- Wakeup capability:
  - MCU wakes up from Low power mode on address detection in slave mode.
- Optional clock stretching

19.3 I^2C general description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I^2C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz), or fast (up to 400 kHz) I^2C bus.

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a STOP generation occurs, allowing Multi-Master capability.

Communication flow

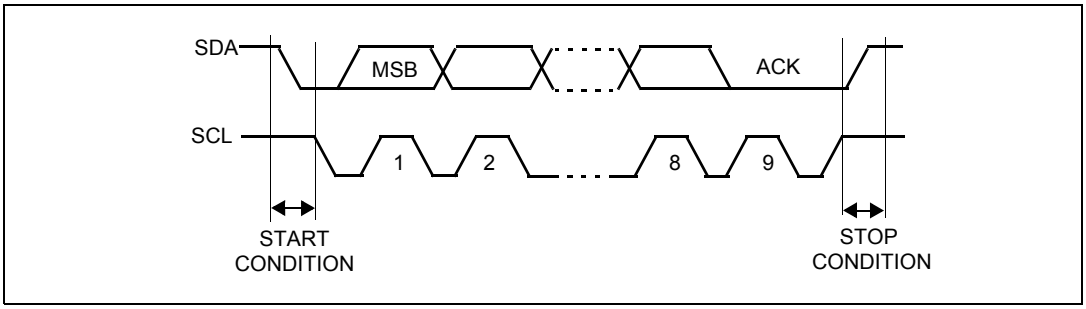
In Master mode, the I^2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7- or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to the following figure.

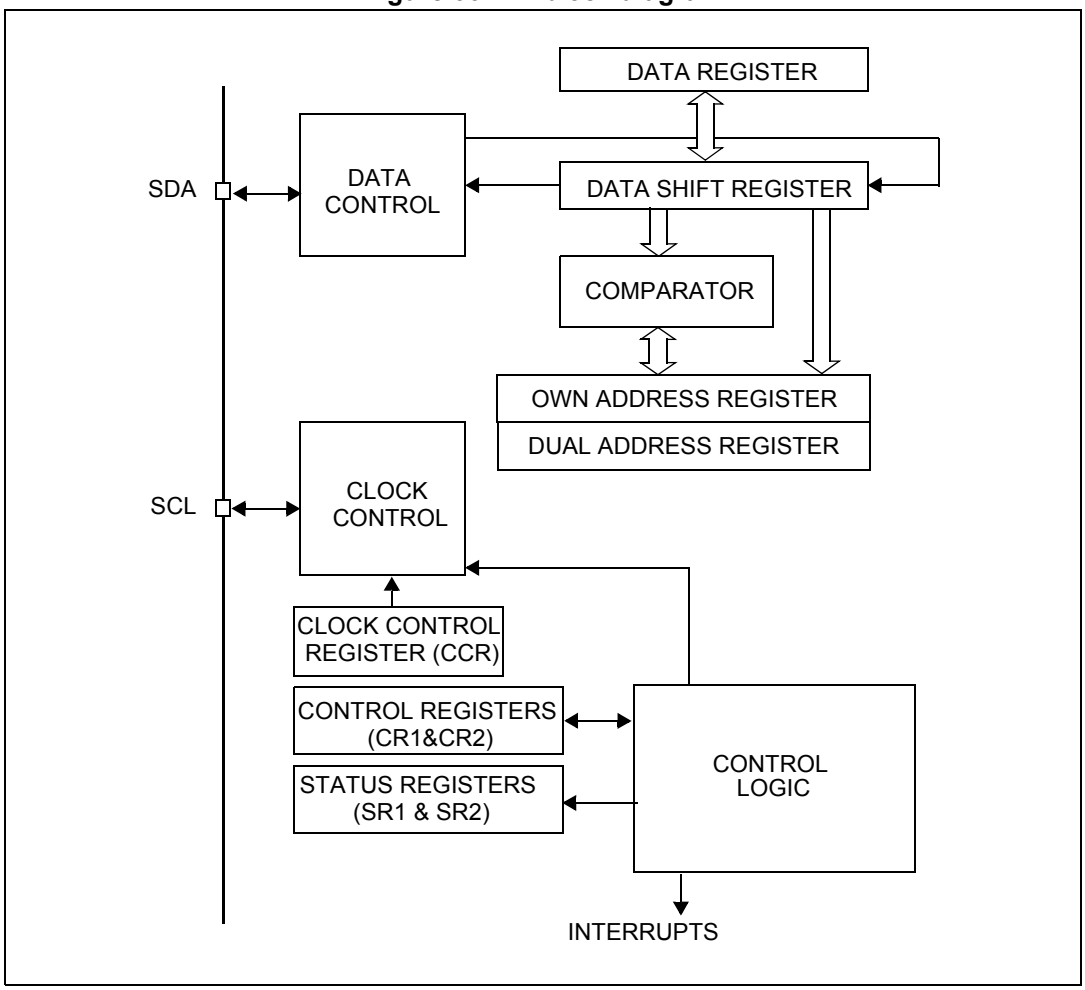
Figure 65. I^2C bus protocol



Acknowledge may be enabled or disabled by software. The I^2C interface addresses (dual addressing, 7-/10-bit and/or general call address) can be selected by software.

The block diagram of the I^2C interface is shown in the following figure.

Figure 66. I^2C block diagram



19.4 I^2C functional description

By default the I^2C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

19.4.1 I^2C slave mode

The peripheral input clock must be programmed in the I2C_FREQR register in order to generate correct timings. The peripheral input clock frequency must be at least:

- 1 MHz in Standard mode
- 4 MHz in Fast mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register. Then it is compared with the address of the interface (OAR1L and OAR2 if ENDUAL = 1) or the General Call address (if ENGC = 1).

Note: In 10-bit addressing mode, the comparison includes the header sequence (11110xx0), where xx denotes the two most significant bits of the address.

Header or address not matched: the interface ignores it and waits for another Start condition.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set and waits for the 8-bit slave address.

Address matched: the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The ADDR bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set. If ENDUAL=1, the software has to read the DUALF bit to check which slave address has been acknowledged.

In 10-bit mode, after receiving the address sequence the slave is always in Receiver mode. It will enter Transmitter mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the slave is in Receiver or Transmitter mode.

Slave transmitter

Following the address reception and after clearing ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

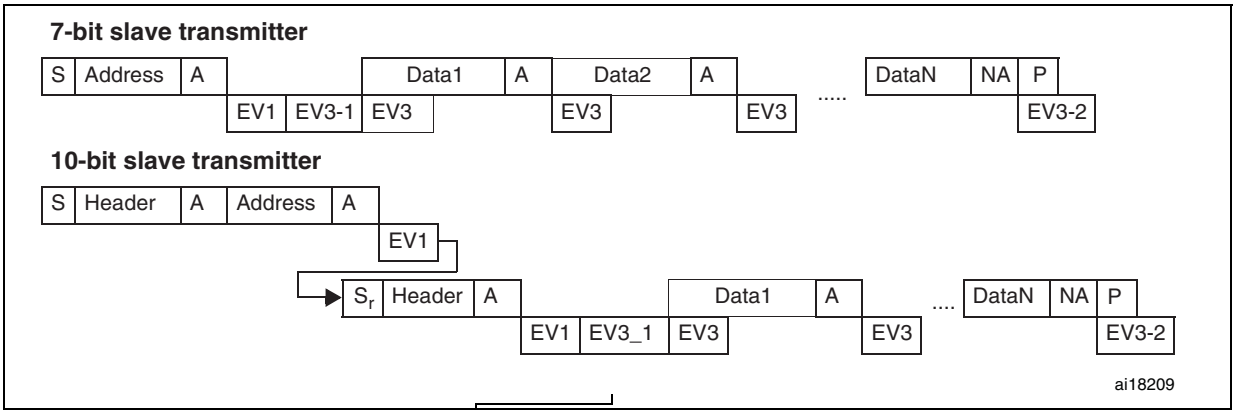
The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see Transfer sequencing EV1 EV3 in the following figure).

When the acknowledge pulse is received:

- The TXE bit is set by hardware with an interrupt if the ITEVTEN and the ITBUFEN bits are set.

If TXE is set and a data was not written in the DR register before the end of the next data transmission, the BTF bit is set and the interface waits until BTF is cleared, by reading the SR1 register and then writing to the DR register, stretching SCL low.

Figure 67. Transfer sequence diagram for slave transmitter



1. Legend:
   S= Start, **S_r** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge, **EVx** = Event (with interrupt if ITEVTEN=1)
   EV1: ADDR =1, cleared by reading SR1 register followed by reading SR3.
   EV3-1: TXE=1, shift register empty, data register empty, write Data1 in DR.
   EV3: TXE=1, shift register not empty, data register empty, cleared by writing DR.
   EV3-2: AF=1, AF is cleared by writing '0' in AF bit of SR2 register.
2. EV1 and EV3-1 events stretch SCL low until the end of the corresponding software sequence.
3. EV3 event stretches SCL low if the software sequence is not completed before the end of the next byte transmission.

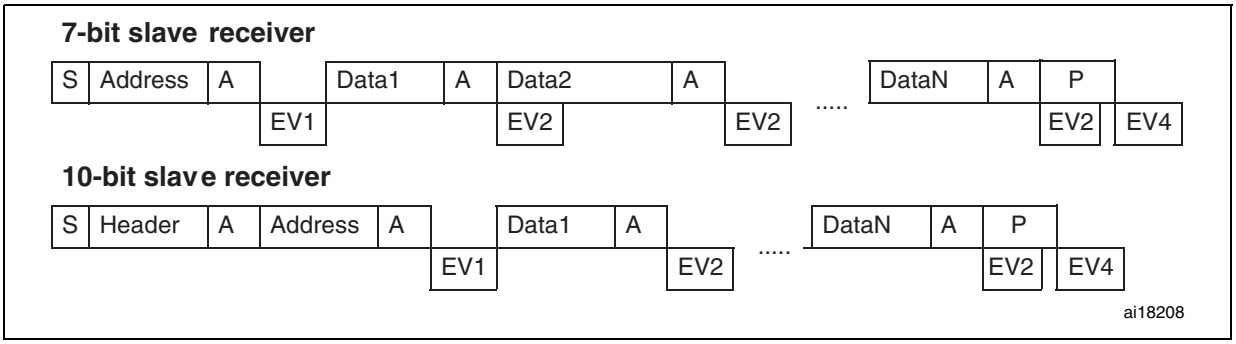
Slave receiver

Following the address reception and after clearing ADDR, the slave receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RXNE bit is set by hardware and an interrupt is generated if the ITEVTEN and ITBUFEN bit is set.

If RXNE is set and the data in the DR register is not read before the end of the next data reception, the BTF bit is set and the interface waits until BTF is cleared, by reading the SR1 register and then reading the DR register, stretching SCL low (see *Figure 68*).

Figure 68. Transfer sequence diagram for slave receiver



1. Legend:
   S= Start, **S_r** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge, **EVx**= Event (with interrupt if ITEVTEN=1)
   EV1: ADDR =1, cleared by reading SR1 register followed by reading SR3.
   EV2: RXNE=1, cleared by reading DR register.
   EV4: STOPF=1, cleared by reading SR1 register followed by writing CR2 register
2. EV1 event stretches SCL low until the end of the corresponding software sequence.
3. EV2 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
4. After checking the SR1 register content, the user should perform the complete clearing sequence for each flag found set. Thus, for the ADDR and STOPF flags, the following sequence is recommended inside the I2C interrupt routine:

   ```
   READ SR1
   if (ADDR == 1) {READ SR1; READ SR3}
   if (STOPF == 1) {READ SR1; WRITE CR2}
   ```

   The purpose is to make sure that both ADDR and STOPF flags are cleared if both are found set.
5. See also: *Note 8 on page 192*.

Closing slave communication

After the last data byte is transferred, a Stop condition is generated by the master. The interface detects this condition and sets the STOPF bit and generates an interrupt if the ITEVTEN bit is set.

STOPF is cleared by a read of the SR1 register followed by a write to the CR2 register (refer to EV4 in *Figure 68: Transfer sequence diagram for slave receiver*).

19.4.2 I²C master mode

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. Master mode is selected as soon as the Start condition is generated on the bus with a START bit.

The following is the required sequence in master mode.

- Program the peripheral input clock in I2C_FREQR Register in order to generate correct timings.
- Configure the clock control registers
- Configure the rise time register
- Program the I2C_CR1 register to enable the peripheral
- Set the START bit in the I2C_CR2 register to generate a Start condition

The peripheral input clock frequency must be at least:

- 1 MHz in Standard mode
- 4 MHz in Fast mode

SCL master clock generation

The CCR bits are used to generate the high and low level of the SCL clock, starting from the generation of the rising and falling edge (respectively). As a slave may stretch the SCL line, the peripheral checks the SCL input from the bus at the end of the time programmed in TRISE bits after the rising edge generation.

- If the SCL line is low, it means that a slave is stretching the bus, and the high level counter stops until the SCL line is detected high. This allows to guarantee the minimum HIGH period of the SCL clock parameter.
- If the SCL line is high, the high level counter keeps on counting.

Indeed, the feedback loop from the SCL rising edge generation by the peripheral to the SCL rising edge detection by the peripheral takes time even if no slave stretches the clock. This loopback duration is linked to SCL rising time (impacting SCL V_{IH} input detection), plus delay due to the analog noise filter present on SCL input path, plus delay due to internal SCL input synchronization with I2C Peripheral clock. The maximum time used by the feedback loop is programmed in TRISE bits, so that the SCL frequency remains stable whatever the SCL rising time.

Start condition

Setting the START bit causes the interface to generate a Start condition and to switch to Master mode (MSL bit set) when the BUSY bit is cleared.

Note: In master mode, setting the START bit causes the interface to generate a Re-Start condition at the end of the current byte transfer.

Once the Start condition is sent:

- The SB bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the Slave address (*Figure 69* & *Figure 70*).

Slave address transmission

Then the slave address is sent to the SDA line via the internal shift register.

- In 10-bit addressing mode, sending the header sequence causes the following event:
  - The ADD10 bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set.

  Then the master waits for a read of the SR1 register followed by a write in the DR register with the second address byte (see *Figure 69* & *Figure 70* Transfer sequencing).

  The ADDR bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set. Then the master waits for a read of the SR1 register followed by a read in the SR3 register (see *Figure 69* & *Figure 70* Transfer sequencing).
- In 7-bit addressing mode, one address byte is sent.

  As soon as the address byte is sent,
  - The ADDR bit is set by hardware and an interrupt is generated if the ITEVTEN bit is set.

  Then the master waits for a read of the SR1 register followed by a read in the SR3 register (see *Figure 69* & *Figure 70* Transfer sequencing).

  The master can decide to enter Transmitter or Receiver mode depending on the LSB of the slave address sent.
- In 7-bit addressing mode,
  - To enter Transmitter mode, a master sends the slave address with LSB reset.
  - To enter Receiver mode, a master sends the slave address with LSB set.
- In 10-bit addressing mode,
  - To enter Transmitter mode, a master sends the header (11110xx0) and then the slave address, (where xx denotes the two most significant bits of the address).
  - To enter Receiver mode, a master sends the header (11110xx0) and then the slave address. Then it should send a repeated Start condition followed by the header (11110xx1), (where xx denotes the two most significant bits of the address).

  The TRA bit indicates whether the master is in Receiver or Transmitter mode.

Master transmitter

Following the address transmission and after clearing ADDR, the master sends bytes from the DR register to the SDA line via the internal shift register.

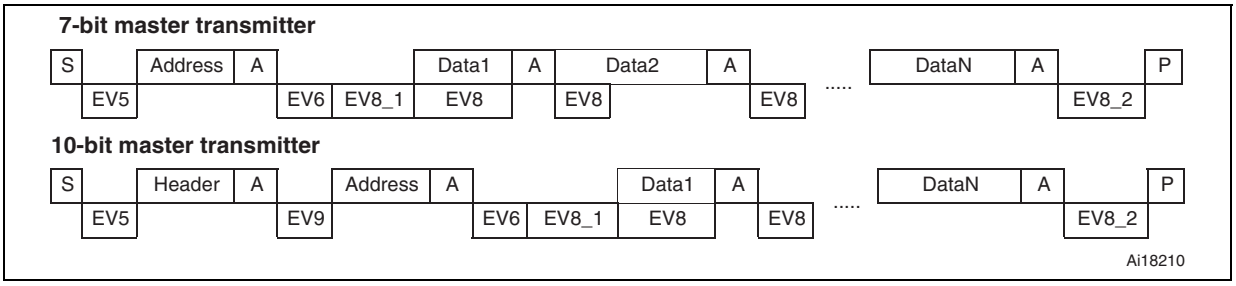
The master waits until the first data byte is written in the DR register, (see *Figure 69* Transfer sequencing EV8_1).

When the acknowledge pulse is received:

- The TXE bit is set by hardware and an interrupt is generated if the ITEVTEN and ITBUFEN bits are set.

If TXE is set and a data byte was not written in the DR register before the end of the next data transmission, BTF is set and the interface waits until BTF is cleared, by reading the SR1 register and then writing to the DR register, stretching SCL low.

Figure 69. Transfer sequence diagram for master transmitter



1. Legend:
   S= Start, **S_r** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
   EVx= Event (with interrupt if ITEVTEN=1)
   EV5: SB=1, cleared by reading SR1 register followed by writing DR register with Address.
   EV6: ADDR=1, cleared by reading SR1 register followed by reading SR3.
   EV8_1: TXE=1, shift register empty, data register empty, write DR register.
   EV8: TXE=1, shift register not empty, data register empty, cleared by writing DR register.
   EV8_2: TXE=1, BTF = 1, Program STOP request. TXE and BTF are cleared by HW by stop condition
   EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register. See also:*Note 8 on page 192*
2. The EV5, EV6, EV9, EV8_1 and EV8_2 events stretch SCL low until the end of the corresponding software sequence.
3. The EV8 event stretches SCL low if the sequence is not complete before the end of the next byte transmission.

Closing the communication

After writing the last byte to the DR register, the STOP bit is set by software to generate a Stop condition (see *Figure 69* Transfer sequencing EV8_2). The interface goes automatically back to slave mode (MSL bit cleared).

Note: Stop condition should be programmed during EV8_2 event, when either TXE or BTF is set.

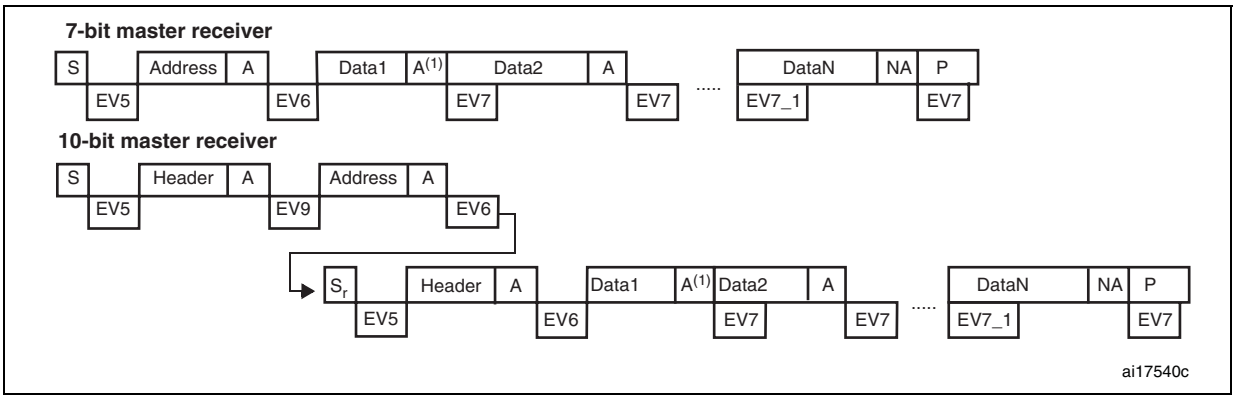
Master receiver

Following the address transmission and after clearing ADDR, the I^2C interface enters Master Receiver mode. In this mode the interface receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RXNE bit is set and an interrupt is generated if the ITEVTEN and ITBUFEN bits are set (*Figure 69* & *Figure 70*).

If the RXNE bit is set and the data in the DR register was not read before the end of the next data reception, the BTF bit is set by hardware and the interface waits for the BTF bit to be cleared by reading I2C_SR1 and then I2C_DR, stretching SCL low.

Figure 70. Transfer sequence diagram for master receiver



1. Legend:
   S= Start, **S_r** = Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge,
   EVx= Event (with interrupt if ITEVTEN=1)
   EV5: SB=1, cleared by reading SR1 register followed by writing DR register.
   EV6: ADDR=1, cleared by reading SR1 register followed by reading SR3. In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.
   EV6_1: no associated flag event, used for 1 byte reception only. Program ACK=0 and STOP=1 after clearing ADDR.
   EV7: RXNE=1, cleared by reading DR register.
   EV7_1: RXNE=1, cleared by reading DR register, program ACK=0 and STOP request
   EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.
2. If a single byte is received, it is NA.
3. The EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
4. The EV7 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
5. The EV7_1 software sequence must be completed before the ACK pulse of the current byte transfer.

Closing the communication

The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a Stop/Re-Start condition.

- In order to generate the non-acknowledge pulse after the last received data byte, the ACK bit must be cleared just after reading the second last data byte (after second last RXNE event).
- In order to generate the Stop/Re-Start condition, software must set the STOP/ START bit just after reading the second last data byte (after the second last RXNE event).
- In case a single byte is to be received, the Acknowledge deactivation and the STOP condition generation are made just after EV6 (in *EV6-1* just after ADDR is cleared).

After the Stop condition generation, the interface goes automatically back to slave mode (MSL bit cleared).

The procedures described below are recommended if the EV7-1 software sequence is not completed before the ACK pulse of the current byte transfer.

These procedures must be followed to make sure that the:

- ACK bit is set low on time, before the end of the last data reception
- STOP bit is set high after the last data reception without reception of supplementary data.

For 2-byte reception:

- Wait until ADDR = 1 (SCL is stretched low until the ADDR flag is cleared).
- Set ACK low and set POS high.
- Clear the ADDR flag.
- Wait until BTF = 1 (data in DR, data 2 in shift register, SCL stretched low until data 1 is read).
- Set STOP high.
- Read data 1 and 2.

For N > 2-byte reception, from N-2 data reception:

- Wait until BTF = 1 (data N-2 in DR, data N-1 in shift register, SCL stretched low until data N-2 is read).
- Set ACK low.
- Read data N-2.
- Wait until BTF = 1 (data N-1 in DR, data N in shift register, SCL stretched low until data N-1 is read).
- Set STOP high.
- Read data N-1 and N.

19.4.3 Error conditions

The following are the error conditions which may cause communication to fail.

Bus error (BERR)

This error occurs when the I2C interface detects an external stop or a start condition during an address or data transfer. In this case:

- The BERR bit is set and an interrupt is generated if the ITERREN bit is set
- In the case of the slave: data are discarded and the lines are released by hardware:
  - In the case of a misplaced start, the slave considers it is a restart and waits for an address or a stop condition.
  - In the case of a misplaced stop, the slave reacts in the same way as for a stop condition and the lines are released by hardware.
- In the case of the master: the lines are not released and there is no effect in the state of the current transmission: software can decide if it wants to abort the current transmission or not.

Acknowledge failure (AF)

This error occurs when the interface detects a non-acknowledge bit. In this case,

- The AF bit is set and an interrupt is generated if the ITERREN bit is set
- A transmitter which receives a NACK must reset the communication:
  - If slave: Lines are released by hardware
  - If master: A stop condition or repeated start must be generated by software

Arbitration lost (ARLO)

This error occurs when the I2C interface detects an arbitration lost condition. In this case,

- The ARLO bit is set by hardware (and an interrupt is generated if the ITERREN bit is set).
- The I2C interface goes automatically back to slave mode (the MSL bit is cleared)
- When the I²C loses the arbitration, it is not able to acknowledge its slave address in the same transfer, but it can acknowledge it after a repeated start from the master.
- Lines are released by hardware

Overrun/underrun error (OVR)

An Overrun error can occur in slave mode when clock stretching is disabled and the I2C interface is receiving data. The interface has received a byte (RXNE = 1) and the data in DR has not been read, before the next byte is received by the interface. In this case,

- The last received byte is lost
- In case of overrun error, software should clear the RXNE bit and the transmitter should re-transmit the last received byte.

Underrun error can occur in slave mode when clock stretching is disabled and the I2C interface is transmitting data. The interface has not updated the DR with the next byte (TXE=1), before the clock comes for the next byte. In this case,

- The same byte in the DR register will be sent again
- The user should make sure that data received on the receiver side during an underrun error is discarded and that the next bytes are written within the clock low time specified in the I^2C bus standard.
- For the first byte to be transmitted, the DR must be written after ADDR is cleared and before the first SCL rising edge. If it is not possible, the receiver must discard the first data.

19.4.4 SDA/SCL line control

- If clock stretching is enabled:
  - Transmitter mode: If TXE = 1 and BTF = 1: the interface holds the clock line low before transmission to wait for the microcontroller to read SR1 and then write the byte in the Data register (both buffer and shift register are empty).
  - Receiver mode: If RXNE = 1 and BTF = 1: the interface holds the clock line low after reception to wait for the microcontroller to read SR1 and then read the byte in the Data Register or write to CR2 (both buffer and shift register are full).
- If clock stretching is disabled in Slave mode:
  - Overrun error in case of RXNE = 1 and no read of DR has been done before the next byte is received. The last received byte is lost.
  - Underrun error in case TXE = 1 and no write into DR has been done before the next byte must be transmitted. The same byte will be sent again.
  - Write Collision not managed.

19.5 I^2C low power modes

Table 38. I^2C interface behavior in low power modes

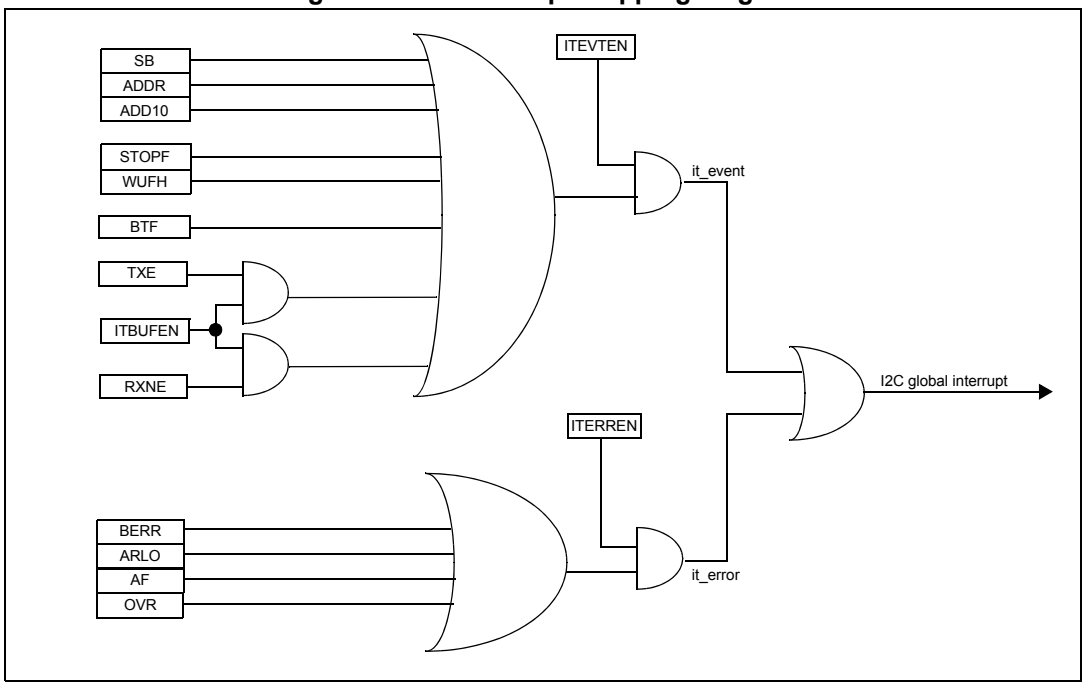
Mode	Description
Wait	No effect on I^2C interface. I^2C interrupts cause the device to exit from Wait mode.
Halt	**In slave mode**: Communication is reset, except for configuration registers. Device is in slave mode. Wakeup from Halt interrupt is generated if ITEVTEN = 1 and address matched (including allowed headers). The matched address is not acknowledged in Halt mode so the master has to send it again when the CPU is woken up to receive an acknowledge. If NOSTRETCH = 0, SCLH will be stretched after acknowledge pulse in Halt mode until WUFH is cleared by software; None of the flags are set by the address which wakes up the CPU. **In master mode**: Communication is frozen until the CPU is woken up. Wakeup from Halt flag and interrupt are generated if ITEVTEN=1 and there is a HALT instruction. *Note: It is forbidden to enter Halt mode while a communication is on going.*

19.6 I^2C interrupts

Table 39. I^2C Interrupt requests

Interrupt event	Event flag	Enable control bit	Exit from Wait	Exit from Halt
Start bit sent (Master)	SB	ITEVTEN	Yes	No
Address sent (Master) or Address matched (Slave)	ADDR			
10-bit header sent (Master)	ADD10			
Stop received (Slave)	STOPF			
Data byte transfer finished	BTF			
Wakeup from Halt	WUFH	ITEVTEN		Yes
Receive buffer not empty	RXNE	ITEVTEN and ITBUFEN		No
Transmit buffer empty	TXE			
Bus error	BERR	ITERREN		
Arbitration loss (Master)	ARLO			
Acknowledge failure	AF			
Overrun/underrun	OVR			

Figure 71. I2C interrupt mapping diagram



19.7 I^2C registers

19.7.1 Control register 1 (I2C_CR1)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
NOSTRETCH	ENGC	Reserved					PE
rw	rw						rw

Bit 7 **NOSTRETCH**: Clock stretching disable (Slave mode)
This bit is used to disable clock stretching in slave mode when ADDR or BTF flag is set, until it is reset by software.
0: Clock stretching enabled
1: Clock stretching disabled

Bit 6 **ENGC**: General call enable
0: General call disabled. Address 0x00 is NACKed.
1: General call enabled. Address 0x00 is ACKed.

Bits 5:1 Reserved

Bit 0 **PE**: Peripheral enable
0: Peripheral disable
1: Peripheral enable: the corresponding I/Os are selected as alternate functions.

Note: If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.
All bit resets due to PE=0 occur at the end of the communication.

19.7.2 Control register 2 (I2C_CR2)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
SWRST	Reserved			POS	ACK	STOP	START
rw				rw	rw	rw	rw

Bit 7 **SWRST**: Software reset

When set, the I2C is at reset state. Before resetting this bit, make sure the I2C lines are released and the bus is free.

0: I2C Peripheral not at reset state

1: I2C Peripheral at reset state

Note: This bit can be used in case the BUSY bit is set to '1' when no stop condition has been detected on the bus.

Bits 6:4 Reserved

Bit 3 **POS**: Acknowledge position (for data reception).

This bit is set and cleared by software and cleared by hardware when PE=0.

0: ACK bit controls the (N)ACK of the current byte being received in the shift register.

1: ACK bit controls the (N)ACK of the next byte which will be received in the shift register.

Note: The POS bit must be used only in 2-byte reception configuration in master mode. It must be configured before data reception starts, as described in the 2-byte reception procedure recommended in Master receiver on page 180.

Bit 2 **ACK**: Acknowledge enable

This bit is set and cleared by software and cleared by hardware when PE=0.

0: No acknowledge returned

1: Acknowledge returned after a byte is received (matched address or data)

Bit 1 **STOP**: Stop generation

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.

– In Master mode:

0: No Stop generation.

1: Stop generation after the current byte transfer or after the current Start condition is sent.

– In Slave mode:

0: No Stop generation.

1: Release the SCL and SDA lines after the current byte transfer.

Bit 0 **START**: Start generation

This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.

– In Master mode:

0: No Start generation

1: Repeated start generation

– In Slave mode:

0: No Start generation

1: Start generation when the bus is free

Note: When STOP or START is set, the user must not perform any write access to I2C_CR2 before the control bit is cleared by hardware. Otherwise, a second STOP START request may occur.

19.7.3 Frequency register (I2C_FREQR)

Address offset: 0x02
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		FREQ[5:0]					
		rw					

Bits 7:6 Reserved

Bits 5:0 **FREQ[5:0]** Peripheral clock frequency. (1)

The FREQ field is used by the peripheral to generate data setup and hold times compliant with the I2C specifications. The FREQ bits must be programmed with the peripheral input clock frequency value:

The allowed range is between 1 MHz and 16 MHz

000000: not allowed

000001: 1 MHz

000010: 2 MHz

...

010000: 16 MHz

Higher values: not allowed

1. The minimum peripheral clock frequencies for respecting the I^2C bus timings are: 1 MHz for standard mode and 4 MHz for fast mode

19.7.4 Own address register 1 LSB (I2C_OAR1L)

Address offset: 0x03
Reset value: 0x00

7	6	5	4	3	2	1	0
ADD1[7:1]							ADD1[0]
rw							rw

Bits 7:1 **ADD1[7:1]** Interface address
bits 7:1 of address

Bit 0 **ADD1[0]** Interface address
7-bit addressing mode: don't care
10-bit addressing mode: bit 0 of address

19.7.5 Own address register 1 MSB (I2C_OAR1H)

Address offset: 0x04
Reset value: 0x00

7	6	5	4	3	2	1	0
ADDMODE	ADDCONF	Reserved			ADD1[9:8]		Reserved
rw	rw				rw		

Bit 7 **ADDMODE** Addressing mode (Slave mode)
0: 7-bit slave address (10-bit address not acknowledged)
1: 10-bit slave address (7-bit address not acknowledged)

Bit 6 **ADDCONF** Address mode configuration
This bit must set by software (must always be written as '1').

Bits 5:3 Reserved

Bits 2:1 **ADD1[9:8]** Interface address
10-bit addressing mode: bits 9:8 of address.

Bit 0 Reserved

19.7.6 Own address register 2 (I2C_OAR2)

Address offset: 0x05
Reset value: 0x00

7	6	5	4	3	2	1	0
ADD2[7:1]							ENDUAL
rw							rw

Bits 7:1 **ADD2[7:1]**: Interface address
Bits 7:1 of address in Dual addressing mode.

Bit 0 **ENDUAL**: Dual addressing mode enable
0: Only OAR1 is recognized in 7-bit Addressing mode.
1: Both OAR1 and OAR2 are recognized in 7-bit Addressing mode.

19.7.7 Data register (I2C_DR)

Address offset: 0x06
Reset value: 0x00

7	6	5	4	3	2	1	0
DR[7:0]							
rw							

Bits 7:0 **DR[7:0]**: Data register (1)(2)(3)
Byte received or to be transmitted to the bus.
– Transmitter mode: Byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in DR once the transmission is started (TXE=1)
– Receiver mode: Received byte is copied into DR (RXNE=1). A continuous transmit stream can be maintained if DR is read before the next data is received (RXNE=1).

1. In slave mode, the address is not copied into DR.
2. Write collision is not managed (DR can be written if TXE=0).
3. If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.

19.7.8 Status register 1 (I2C_SR1)

Address offset: 0x07
Reset value: 0x00

7	6	5	4	3	2	1	0
TXE	RXNE	Reserved	STOPF	ADD10	BTF	ADDR	SB
r	r		r	r	r	r	r

Bit 7 **TXE**: Data register empty (transmitters) (1)

0: Data register not empty
1: Data register empty

– Set when DR is empty in transmission. TXE is not set during address phase.
– Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

Note: TXE cannot be cleared by writing the first data in transmission or by writing a data when the BTF bit is set as in both cases, the DR register is still empty.

Bit 6 **RXNE**: Data register not empty (receivers) (2) (3)

0: Data register empty
1: Data register not empty

– Set when data register is not empty in receiver mode. RXNE is not set during address phase.
– Cleared by software reading or writing the DR register or by hardware when PE=0.

Note: RXE cannot be cleared by reading a data when the BTF bit is set as the DR register is still full in this case.

Bit 5 Reserved

Bit 4 **STOPF**: Stop detection (Slave mode) (4)(5)

0: No Stop condition detected
1: Stop condition detected

– Set by hardware when a Stop condition is detected on the bus by the slave after an acknowledge (if ACK=1).
– Cleared by software reading the SR1 register followed by a write in the CR2 register, or by hardware when PE=0

Bit 3 **ADD10**: 10-bit header sent (Master mode) (6)

0: No ADD10 event occurred.
1: Master has sent first address byte (header).

– Set by hardware when the master has sent the first byte in 10-bit address mode.
– Cleared by software reading the SR1 register followed by a write in the DR register of the second address byte, or by hardware when PE=0.

Bit 2 **BTF**: Byte transfer finished (7)(8)

0: Data byte transfer not done
1: Data byte transfer succeeded

– Set by hardware when NOSTRETCH=0 and:
  – In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RXNE=1).
  – In transmission when a new byte should be sent and DR has not been written yet (TXE=1).
– Cleared by software reading SR1 followed by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE=0.

Bit 1 **ADDR**: Address sent (master mode)/matched (slave mode) (8)(9)

This bit is cleared by software reading SR1 register followed reading SR3, or by hardware when PE=0.

– Address matched (Slave)

0: Address mismatched or not received.

1: Received address matched.

– Set by hardware as soon as the received slave address matched with the OAR registers content or a general call or a SMBus is recognized. (when enabled depending on configuration).

– Address sent (Master)

0: No end of address transmission

1: End of address transmission

– For 10-bit addressing, the bit is set after the ACK of the 2nd byte.

– For 7-bit addressing, the bit is set after the ACK of the byte.

Note: ADDR is not set after a NACK reception

Bit 0 **SB**: Start bit (Master mode) (8)

0: No Start condition

1: Start condition generated.

– Set when a Start condition generated.

– Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

1. The interrupt will be generated when DR is copied into shift register after an ACK pulse. If a NACK is received, copy is not done and TXE is not set.
2. The interrupt will be generated when Shift register is copied into DR after an ACK pulse.
3. RXNE is not set in case of ARLO event.
4. The STOPF bit is not set after a NACK reception.
5. It is recommended to perform the complete clearing sequence (READ SR1 then WRITE CR2) after STOPF is set. Refer to *Figure 68: Transfer sequence diagram for slave receiver on page 176*
6. The ADD10 bit is not set after a NACK reception.
7. The BTF bit is not set after a NACK reception, or in case of an ARLO event.
8. Due to timing constraints, when in standard mode if CCR is less than 9 (i.e. with peripheral clock below 2 MHz) with $f_{MASTER} = f_{CPU}$ and the event interrupt disabled, the following procedure must be followed:
modify the reset sequence in order to insert at least 5 cycles between each operations in the flag clearing sequence. For example, when $f_{MASTER} = f_{CPU}$ = 1 MHz, use the following sequence to poll the SB bit:
```
_label_wait: BTJF I2C_SR1,SB,_label_wait
NOP ;
NOP;
NOP ;
NOP
NOP
LD I2C_DR, A ; once executed, the SB bit is then cleared.
```
9. In slave mode, it is recommended to perform the complete clearing sequence (READ SR1 then READ SR3) after ADDR is set. Refer to *Figure 68: Transfer sequence diagram for slave receiver on page 176*.

19.7.9 Status register 2 (I2C_SR2)

Address offset: 0x08
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		WUFH	Reserved	OVR	AF	ARLO	BERR
		rc_w0		rc_w0	rc_w0	rc_w0	rc_w0

Bits 7:6 Reserved

Bit 5 **WUFH**: Wakeup from Halt

0: no wakeup from Halt mode

1: 7-bit address or header match in Halt mode (slave mode) or Halt entered when in master mode.

Note: This bit is set asynchronously in slave mode (during HALT mode). It is set only if ITEVTEN = 1.

– cleared by software writing 0, or by hardware when PE=0.

Bit 4 Reserved

Bit 3 **OVR**: Overrun/underrun

0: No overrun/underrun

1: Overrun or underrun

– Set by hardware in slave mode when NOSTRETCH=1 and:

– In reception when a new byte is received (including ACK pulse) and the DR register has not been read yet. New received byte is lost.

– In transmission when a new byte should be sent and the DR register has not been written yet. The same byte is sent twice.

Cleared by software writing 0, or by hardware when PE=0.

Note: if the DR write occurs very close to the SCL rising edge, the sent data is unspecified and a hold timing error occurs.

Bit 2 **AF**: Acknowledge failure.

0: No acknowledge failure

1: Acknowledge failure

– Set by hardware when no acknowledge is returned.

– Cleared by software writing 0, or by hardware when PE=0.

Bit 1 **ARLO**: Arbitration lost (master mode)

0: No Arbitration lost detected

1: Arbitration lost detected

Set by hardware when the interface loses the arbitration of the bus to another master.

– Cleared by software writing 0, or by hardware when PE=0.

After an ARLO event the interface switches back automatically to Slave mode (MSL=0).

Bit 0 **BERR**: Bus error

0: No misplaced Start or Stop condition

1: Misplaced Start or Stop condition

– Set by hardware when the interface detects a SDA rising or falling edge while SCL is high, occurring in a non-valid position during a byte transfer.

– Cleared by software writing 0, or by hardware when PE=0.

19.7.10 Status register 3 (I2C_SR3)

Address offset: 0x09
Reset value: 0x00

7	6	5	4	3	2	1	0
DUALF	Reserved		GENCALL	Reserved	TRA	BUSY	MSL
r			r		r	r	r

Bit 7 **DUALF**: Dual flag (Slave mode)

0: Received address matched with OAR1
1: Received address matched with OAR2

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bits 6:5 Reserved

Bit 4 **GENCALL**: General call header (Slave mode)

0: No general call
1: General call header received when ENGC=1

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 3 Reserved

Bit 2 **TRA**: Transmitter/Receiver

0: Data bytes received
1: Data bytes transmitted

This bit is set depending on R/W bit of address byte, at the end of total address phase.
It is also cleared by hardware after detection of Stop condition (STOPF=1), repeated Start condition, loss of bus arbitration (ARLO=1), or when PE=0.

Bit 1 **BUSY**: Bus busy

0: No communication on the bus
1: Communication ongoing on the bus

– Set by hardware on detection of SDA or SCL low
– cleared by hardware on detection of a Stop condition.

It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE=0).

Bit 0 **MSL**: Master/Slave

0: Slave mode
1: Master mode

– Set by hardware as soon as the interface is in Master mode (SB=1).
– Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

Note: Reading I2C_SR3 after reading I2C_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I2C_SR1. Consequently, I2C_SR3 must be read only when ADDR is found set in I2C_SR1 or when the STOPF bit is cleared.

19.7.11 Interrupt register (I2C_ITR)

Address offset: 0x0A
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved					ITBUFEN	ITEVTEN	ITERREN
					rw	rw	rw

Bits 7:3 Reserved

Bit 2 **ITBUFEN**: Buffer interrupt enable
0: TXE = 1 or RXNE = 1 does not generate any interrupt.
1:TXE = 1 or RXNE = 1 generates Event interrupt.

Bit 1 **ITEVTEN**: Event interrupt enable
0: Event interrupt disabled
1: Event interrupt enabled
This interrupt is generated when:
– SB = 1 (Master)
– ADDR = 1 (Master/Slave)
– ADD10= 1 (Master)
– STOPF = 1 (Slave)
– BTF = 1 with no TXE or RXNE event
– TXE event to 1 if ITBUFEN = 1
– RXNE event to 1if ITBUFEN = 1
– WUFH = 1 (asynchronous interrupt to wakeup from Halt)

Bit 0 **ITERREN**: Error interrupt enable
0: Error interrupt disabled
1: Error interrupt enabled
– This interrupt is generated when:
– BERR = 1
– ARLO = 1
– AF = 1
– OVR = 1

19.7.12 Clock control register low (I2C_CCRL)

Address offset: 0x02
Reset value: 0x0B

7	6	5	4	3	2	1	0
CCR[7:0]							
rw							

Bits 7:0 **CCR[7:0]** Clock control register (Master mode)

Controls the SCLH clock in Master mode.

– Standard mode:

Period(I2C) = 2 * CCR * t_{MASTER}

t_{high} = CCR * t_{MASTER}

t_{low} = CCR * t_{MASTER}

– Fast mode:

If DUTY = 0:

Period(I2C) = 3* CCR * t_{MASTER}

t_{high} = CCR * t_{MASTER}

t_{low} = 2 * CCR * t_{MASTER}

If DUTY = 1: (to reach 400 kHz)

Period(I2C) = 25 * CCR * t_{MASTER}

t_{high} = 9 * CCR * t_{MASTER}

t_{low} = 16 * CCR * t_{MASTER}

Note: $t_{CK} = 1/ f_{MASTER}$. f_{MASTER} is the input clock to the peripheral configured using clock control register.

The minimum allowed value is 04h, except in FAST DUTY mode where the minimum allowed value is 0x01.

$t_{high} = t_{r(SCL)} + t_{w(SCLH)}$. See device datasheet for the definitions of parameters.

$t_{low} = t_{f(SCL)} + t_{w(SCLL)}$. See device datasheet for the definitions of parameters.

I2C communication speed, $f_{SCL} = 1/(t_{high} + t_{low})$

The real frequency may differ due to the analog noise filter input delay.

19.7.13 Clock control register high (I2C_CCRH)

Address offset: 0x0C
Reset value: 0x00

7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:8]			
rw	rw			rw			

Bit 7 **F/S**: I2C master mode selection
0: Standard mode I2C
1: Fast mode I2C

Bit 6 **DUTY**: Fast mode duty cycle
0: Fast mode t_{low}/t_{high} = 2
1: Fast mode t_{low}/t_{high} = 16/9 (see CCR)

Bits 5:4 Reserved

Bits 3:0 **CCR[11:8]**: Clock control register in Fast/Standard mode (Master mode)(1)
Controls the SCLH clock in master mode.

– Standard mode:
Period(I2C) = 2 * CCR * t_{MASTER}
t_{high} = CCR * t_{MASTER}
t_{low} = CCR * t_{MASTER}

– Fast mode:
If DUTY = 0:
Period(I2C) = 3 * CCR * t_{MASTER}
t_{high} = CCR * t_{MASTER}
t_{low} = 2 * CCR * t_{MASTER}
If DUTY = 1: (to reach 400 kHz)
Period(I2C) = 25 * CCR * t_{MASTER}
t_{high} = 9 * CCR * t_{MASTER}
t_{low} = 16 * CCR * t_{MASTER}
For instance: in standard mode, to generate a 100 kHz SCL frequency:
If FREQR = 08, t_{MASTER} = 125 ns so CCR must be programmed with 0x28
(0x28 <=> 40 x 125 ns = 5000 ns.)

Note: $t_{high} = t_{r(SCL)} + t_{w(SCLH)}$. See device datasheet for the definitions of parameters
$t_{low} = t_{f(SCL)} + t_{w(SCLL)}$. See device datasheet for the definitions of parameters
The real frequency may differ due to the analog noise filter input delay.

1. Refer to *Table 40: I2C_CCR values for SCL frequency (fMASTER = 16 MHz) on page 198*

Note: The CCR registers must be configured only when the I²C is disabled (PE=0).
f_{MASTER} = multiple of 10 MHz is required to generate Fast clock at 400 kHz.
$f_{MASTER} \geq 1$ MHz is required to generate Standard clock at 100 kHz.

Table 40. I2C_CCR values for SCL frequency (f_{MASTER} = 16 MHz[1])

I^2C speed	I^2C frequency (f_{SCL}) Hz	f_{MASTER} = 16 MHz			
		Actual (Hz)	% error (%)	I2C_CCR (h)	Duty cycle bit
Fast speed	400000	410256.41	2.56	D	0
	370000	380952.38	2.96	E	0
	350000	355555.56	1.59	F	0
	320000	320000	0	2	1
	300000	313725.49	4.57	11	0
	270000	280701.75	3.96	13	0
	250000	253968.25	1.59	15	0
	220000	222222.22	1.01	18	0
	200000	205128.20	2.56	1A	0
	170000	172043.01	1.20	1F	0
	150000	152380.95	1.59	23	0
	120000	121212.12	1.01	2C	0
Standard speed	100000	100000	0	50	No impact
	50000	50000	0	A0	
	30000	30075.19	0.25	10A	
	20000	20000	0	190	

1. The following table gives the values to be written in the I2C_CCR register to obtain the required I²C SCL line frequency

19.7.14 TRISE register (I2C_TRISER)

Address offset: 0x0D

Reset value: 0x02

7	6	5	4	3	2	1	0
Reserved		TRISE[5:0]					
		rw					

Bits 7:6 Reserved

Bits 5:0 **TRISE[5:0]** Maximum rise time in Fast/Standard mode (Master mode)

These bits should provide the maximum duration of the SCL feedback loop in master mode. The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration.

These bits must be programmed with the maximum SCL rise time given in the I2C bus specification, incremented by 1.

For instance: in standard mode, the maximum allowed SCL rise time is 1000 ns.

If the value in the I2C_FREQR register = 08h, then t_{MASTER} = 125 ns therefore the TRISE[5:0] bits must be programmed with 0x09.

(1000 ns / 125 ns = 8 + 1)

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Note: TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).

19.7.15 I^2C register map and reset values

Table 41. I^2C register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	I2C_CR1 Reset value	NO STRETCH 0	ENGC 0	- 0	- 0	- 0	- 0	- 0	PE 0
0x01	I2C_CR2 Reset value	SWRST 0	- 0	- 0	- 0	POS 0	ACK 0	STOP 0	START 0
0x02	I2C_FREQR Reset value	- 0	- 0	FREQ[5:0] 000000					
0x03	I2C_OAR1L Reset value	ADD1[7:1] 0000000							ADD1[0] 0
0x04	I2C_OAR1H Reset value	ADDMODE 0	ADDCONF 0	- 0	- 0	- 0	ADD1[9:8] 00		- 0
0x05	I2C_OAR2 Reset value	ADD2[7:1] 0							ENDUAL 0
0x06	I2C_DR Reset value	DR[7:0] 0							
0x07	I2C_SR1 Reset value	TXE 0	RXNE 0	- 0	STOPF 0	ADD10 0	BTF 0	ADDR 0	SB 0
0x08	I2C_SR2 Reset value	- 0	- 0	WUFH 0	- 0	OVR 0	AF 0	ARLO 0	BERR 0
0x09	I2C_SR3 Reset value	DUALF 0	- 0	- 0	GENCALL 0	- 0	TRA 0	BUSY 0	MSL 0

Table 41. I^2C register map (continued)

Address offset	Register name	7	6	5	4	3	2	1	0
0x0A	I2C_ITR Reset value	- 0	- 0	- 0	- 0	- 0	ITBUFEN 0	ITEVTEN 0	ITERREN 0
0x0B	I2C_CCRL Reset value	CCR[7:0] 00000000							
0x0C	I2C_CCRH Reset value	FS 0	DUTY 0	- 0	- 0	CCR[11:8] 0000			
0x0D	I2C_TRISER Reset value	- 0	- 0	TRISE[5:0] 000010					

20 Serial peripheral interface (SPI)

20.1 Introduction

The serial peripheral interface (SPI) allows half/ full duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multi-master configuration.

20.2 SPI main features

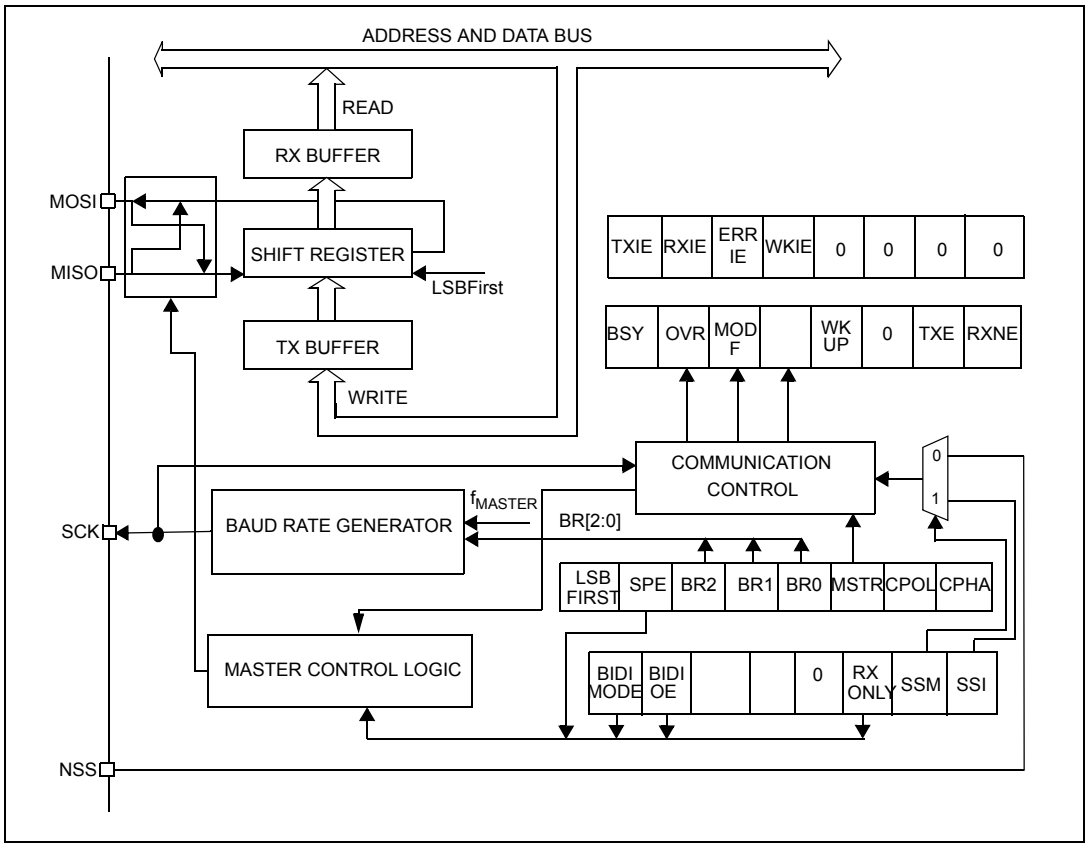
- Full duplex synchronous transfers (on 3 lines)
- Simplex synchronous transfers on 2 lines with or without a bidirectional data line
- Master or slave operation
- 8 Master mode frequencies (f_{MASTER}/2 max.)
- Slave mode frequency (f_{MASTER}/2 max.)
- Faster communication - Maximum SPI speed: 8 MHz
- NSS management by hardware or software for both master and slave
- 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
- Master mode fault and overrun flags with interrupt capability
- Wakeup capability:
  The MCU wakes up from Low power mode in full or half duplex transmit-only modes

20.3 SPI functional description

20.3.1 General description

The block diagram of the SPI is shown in *Figure 72*.

Figure 72. SPI block diagram



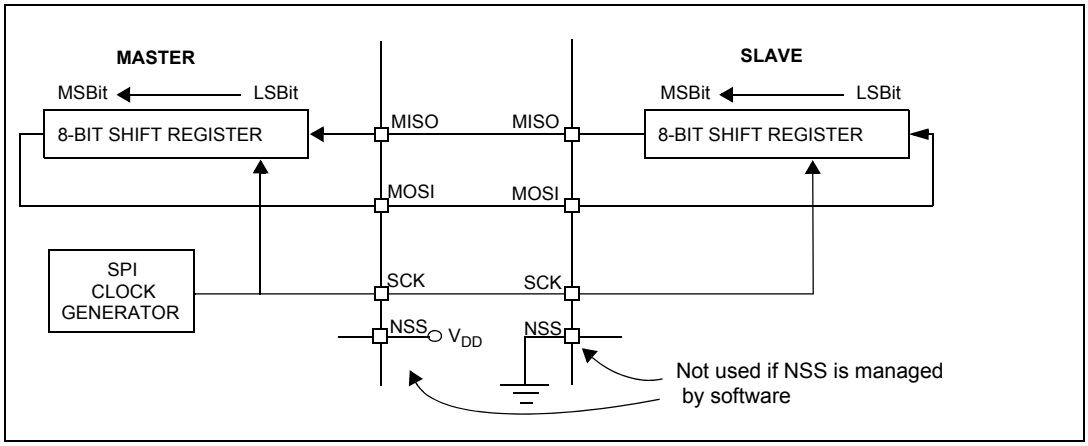
The SPI is connected to external devices through four pins:

- MISO: Master In / Slave Out data (port C7). This pin can be used to transmit data in slave mode and receive data in master mode.
- MOSI: Master Out / Slave In data (port C6). This pin can be used to transmit data in master mode and receive data in slave mode.
- SCK: Serial Clock output (port C5) for SPI masters and Serial Clock input for SPI slaves.
- NSS: Slave select (port E5). This is a optional pin to select a slave device. This pin acts as a 'chip select' to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave NSS inputs can be driven by standard I/O ports on the master device. When configured in master mode (MSTR bit =1) and if NSS is pulled low, the SPI enters master mode fault state: the MSTR bit is automatically reset and the device is configured in slave mode (refer to *Section 20.3.8: Error flags on page 216*).

A basic example of interconnections between a single master and a single slave is illustrated in *Figure 73*.

Note: When using the SPI in High-speed mode, the I/Os where SPI outputs are connected should be programmed as fast slope outputs in order to be able to reach the expected bus speed.

Figure 73. Single master/ single slave application



The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via MOSI pin, the slave device responds the MISO pin. This implies full duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

Slave select (NSS) pin management

A hardware or software slave select management configuration can be set using the Software slave select management (SSM) bit from the SPI_CR2 register.

- **Software NSS management (SSM = 1):** with this configuration, slave select information is driven internally by the Internal slave select (SSI) bit value in the SPI_CR2 register.The external NSS pin remains free for other application uses.
- **Hardware NSS management (SSM = 0):** For devices set as master, this configuration allows multimaster capability. For devices set as slave, the NSS pin works as a classical NSS input. The slave is selected when the NSS line is in low level and is not selected if the NSS line is in high level.

Note: When the master is communicating with SPI slaves which need to be deselected between transmissions, the NSS pin must be configured as a GPIO.

Clock phase and clock polarity

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits. The CPOL (clock polarity) bit controls the steady state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, SCK pin has a low level idle state. If CPOL is set, SCK pin has a high level idle state.

Note: Make sure the SPI pin is configured at the idle state level of the SPI in order to avoid generating an edge on the SPI clock pin when enabling or disabling the SPI cell.

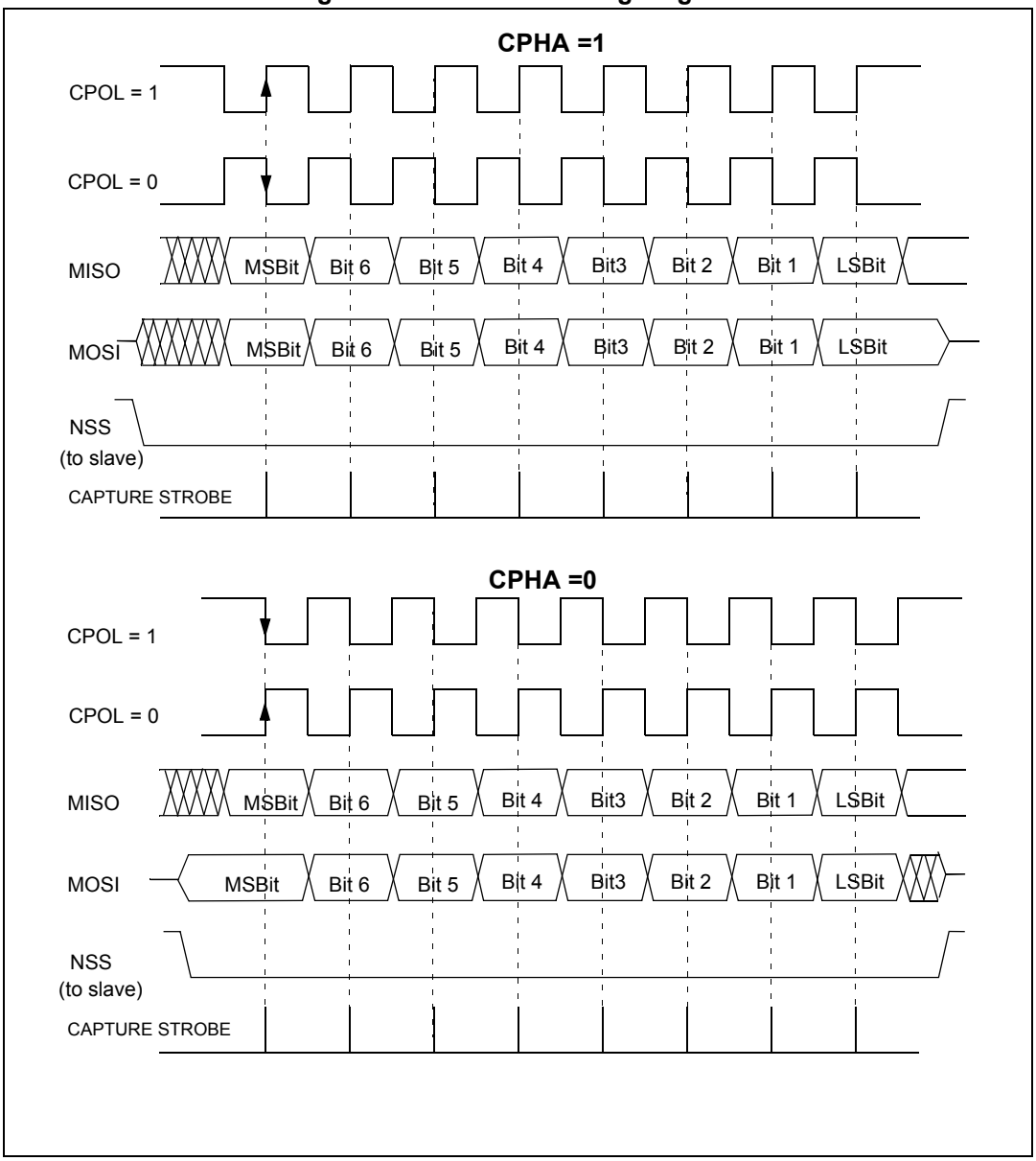
If CPHA (clock phase) bit is set, the second edge on the SCK pin (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set) is the MSBit capture strobe. Data is latched on the occurrence of the first clock transition. If CPHA bit is reset, the first edge on the SCK pin (falling edge if CPOL bit is set, rising edge if CPOL bit is reset) is the MSBit capture strobe. Data is latched on the occurrence of the second clock transition.

The combination of the CPOL clock polarity and CPHA (clock phase) bits selects the data capture clock edge.

Figure 74 shows an SPI transfer with the four combinations of the CPHA and CPOL bits. The diagram may be interpreted as a master or slave timing diagram where the SCK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.

Note:
1. *Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.*
2. *Master and slave must be programmed with the same timing mode.*
3. *The idle state of SCK must correspond to the polarity selected in the SPI_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).*

Figure 74. Data clock timing diagram



1. These timings are shown with the LSBFIRST bit reset in the SPI_CR1 register.

Frame format

Data can be shifted out either MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 Register.

20.3.2 Configuring the SPI in slave mode

In slave configuration, the serial clock is received on the SCK pin from the master device. The value set in the BR[2:0] bits in the SPI_CR1 register, does not affect the data transfer rate.

Follow the procedure below to configure the SPI in slave mode:

1. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see *Figure 74*). For correct data transfer, the CPOL and CPHA bits must be configured in the same way in the slave device and the master device.
2. The frame format (MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 register) must be the same as the master device.
3. In Hardware mode (refer to *Slave select (NSS) pin management on page 203*), the NSS pin must be connected to a low level signal during the complete data transmit sequence. In NSS Software mode, set the SSM bit and clear the SSI bit in the SPI_CR2 register.
4. Clear the MSTR bit and set the SPE bit to assign the pins to alternate functions.

In this configuration the MOSI pin is a data input and the MISO pin is a data output.

Note: In applications with a parallel multi-slave structure, with separate NSS signals and the slave MISO outputs connected together, the corresponding GPIO registers must be configured correctly. The SPI_MISO pin is controlled by the SPI peripheral only when the NSS signal is active and the device is selected as slave. When the NSS signal is released, the pin is driven by GPIO register settings only. To function correctly, the GPIO has to be configured in input pull-up mode with no interrupt. This configuration is done using the GPIO_DDR, GPIO_CR1 and GPIO_CR2 registers - see Section 10.8.1: Alternate function output.

20.3.3 Configuring the SPI master mode

In a master configuration, the serial clock is generated on the SCK pin.

Follow the procedure below to configure the SPI in master mode:

1. Select the BR[2:0] bits to define the serial clock baud rate (see SPI_CR1 register).
2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see *Figure 74*).
3. Configure the LSBFIRST bit in the SPI_CR1 register to define the frame format.
4. In Hardware mode, connect the NSS pin to a high-level signal during the complete data transmit sequence. In software mode, set the SSM and SSI bits in the SPI_CR2 register.
5. Set the MSTR and SPE bits (they remain set only if the NSS pin is connected to a high-level signal).

In this configuration the MOSI pin is a data output and to the MISO pin is a data input.

20.3.4 Configuring the SPI for simplex communications

The SPI is capable of operating in simplex mode in 2 configurations.

- 1 clock and 1 bidirectional data wire
- 1 clock and 1 data wire (Receive-only or Transmit-only)

1 clock and 1 bidirectional data wire

This mode is enabled by setting the BDM bit in the SPI_CR2 register. In this mode SCK is used for the clock, and MOSI in master or MISO in slave mode is used for data communication. The transfer direction (Input/output) is selected by the BDOE bit in the SPI_CR2 register. When this bit is set to 1, the data line is output, otherwise it is input.

1 clock and 1 unidirectional data wire (BDM = 0)

In this mode, the application can use the SPI either in transmit-only mode or in receive-only mode:

- Transmit-only mode is similar to full-duplex mode (BDM = 0, RXONLY = 0): the data is transmitted to the transmit pin (MOSI in master mode or MISO in slave mode) and the receive pin (MISO in master mode or MOSI in slave mode) can be used as general purpose I/O. In this case, the application just needs to ignore the Rx buffer (if the data register is read, it does not contain the received value).
- In receive-only mode, the application can disable the SPI output function by setting the RXONLY bit in the SPI_CR2 register. In this case, it frees the transmit I/O pin (MOSI in master mode or MISO in slave mode) so it can be used for other purposes.

To start the communication in receive-only mode, configure and enable the SPI:

- In master mode, the communication starts immediately and stops when the SPE bit is reset and the current reception stops. There is no need to read the BSY flag in this mode. It is always set when an SPI communication is ongoing.
- In slave mode, the SPI continues to receive as long as the NSS is pulled down (or the SSI bit is reset in NSS software mode) and the SCK is running.

20.3.5 Data transmission and reception procedures

Rx and Tx buffer

In reception, data are received and then stored into an internal Rx buffer while In transmission, data are first stored into an internal Tx buffer before being transmitted.

A read access of the SPI_DR register returns the Rx buffered value whereas a write access of the SPI_DR stores the written data into the Tx buffer.

Start sequence in master mode

- In full-duplex (BDM = 0 and RXONLY = 0)
  - The sequence begins when data is written into the SPI_DR register (Tx buffer).
  - The data is then parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MOSI pin.
  - At the same time, the received data on MISO pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).

- In unidirectional receive-only mode (BDM = 0 and RXONLY = 1)
  - The sequence begins as soon as the bit SPE = 1
  - Only the receiver is activated and the received data on MISO pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).
- In bidirectional mode, when transmitting (BDM = 1 and BDOE = 1)
  - The sequence begins when a data is written into the SPI_DR register (Tx buffer).
  - The data is then parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MOSI pin.
  - No data is received.
- In bidirectional mode, when receiving (BDM = 1 and BDOE = 0)
  - The sequence begins as soon as SPE = 1 and BDOE = 0.
  - The received data on MOSI pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).
  - The transmitter is not activated and no data is shifted out serially to the MOSI pin.

Start sequence in slave mode

- In full-duplex (BDM=0 and RXONLY=0)
  - The sequence begins when the slave device receives the clock signal and the first bit of the data on its MOSI pin. The remaining 7 bits are loaded into the shift register.
  - At the same time, the data is parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MISO pin. The software must have written the data to be sent before the SPI master device initiates the transfer.
- In unidirectional receive-only mode (BDM = 0 and RXONLY = 1)
  - The sequence begins when the slave device receives the clock signal and the first bit of the data on its MOSI pin. The remaining 7 bits are loaded into the shift register.
  - The transmitter is not activated and no data is shifted out serially to the MISO pin.
- In bidirectional mode, when transmitting (BDM = 1 and BDOE = 1)
  - The sequence begins when the slave device receives the clock signal and the first bit of the Tx buffer is transmitted to the MISO pin.
  - The data is then parallel loaded from the Tx buffer into the 8-bit shift register during the first bit transmission and then shifted out serially to the MISO pin. The software must have written the data to be sent before the SPI master device starts the transfer.
  - no data is received.
- In bidirectional mode, when receiving (BDM = 1 and BDOE = 0)
  - The sequence starts when the slave device receives the clock signal and the first bit of the data to its MISO pin.
  - The data received on MISO pin is shifted in serially to the 8-bit shift register and then parallel loaded into the SPI_DR register (Rx Buffer).
  - The transmitter is not activated and no data is shifted out serially to the MISO pin.

Handling data transmission and reception

The TXE flag (Tx buffer empty) is set when the data is transferred from the Tx buffer to the shift register. It indicates that the internal Tx buffer is ready to be loaded with the next data. An interrupt can be generated if TXIE bit in the SPI_ICR register is set.

Note: The software must ensure that TXE flag is set to 1 before attempting to write into the Tx buffer. Otherwise, it will overwrite the data which was previously written in the Tx buffer.

The RXNE flag (Rx buffer not empty) is set on the last sampling clock edge, when the data is transferred from the shift register to the Rx buffer. It indicates that a data is ready to be read from the SPI_DR register. An interrupt can be generated if RXIE bit in the SPI_ICR register is set. Clearing the RXNE bit is performed by reading the SPI_DR register.

In some configurations, the BSY flag can be used during the last data transfer to wait until the completion of the transfer.

Full Duplex Transmit and receive procedure in master or slave mode (BDM=0 and RXONLY = 0)

1. Enable the SPI by setting the SPE bit
2. Write the first data to be transmitted in the SPI_DR register (this clears the TXE flag).
3. Wait until TXE = 1 and write the second data to be transmitted. Then wait until RXNE = 1 and read the SPI_DR to get the first received data (this clears the RXNE bit). Repeat this operation for each data to be transmitted/received until the n-1 received data.
4. Wait until RXNE = 1 and read the last received data.
5. Wait until TXE = 1 and then wait until BSY = 0 before disabling the SPI.

This procedure can also be implemented using dedicated interrupt subroutines launched at each rising edge of RXNE or TXE flags.

Figure 75. TXE/RXNE/BSY behavior in full duplex mode (RXONLY = 0). Case of continuous transfers

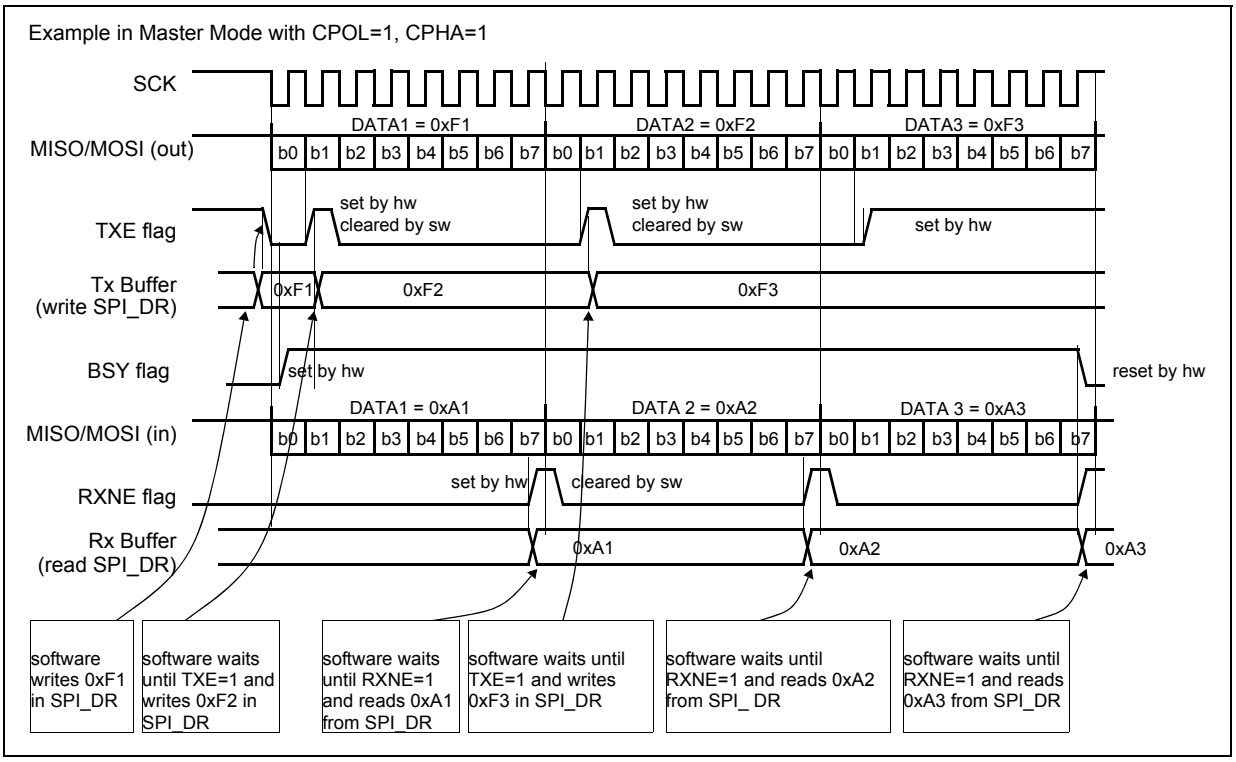
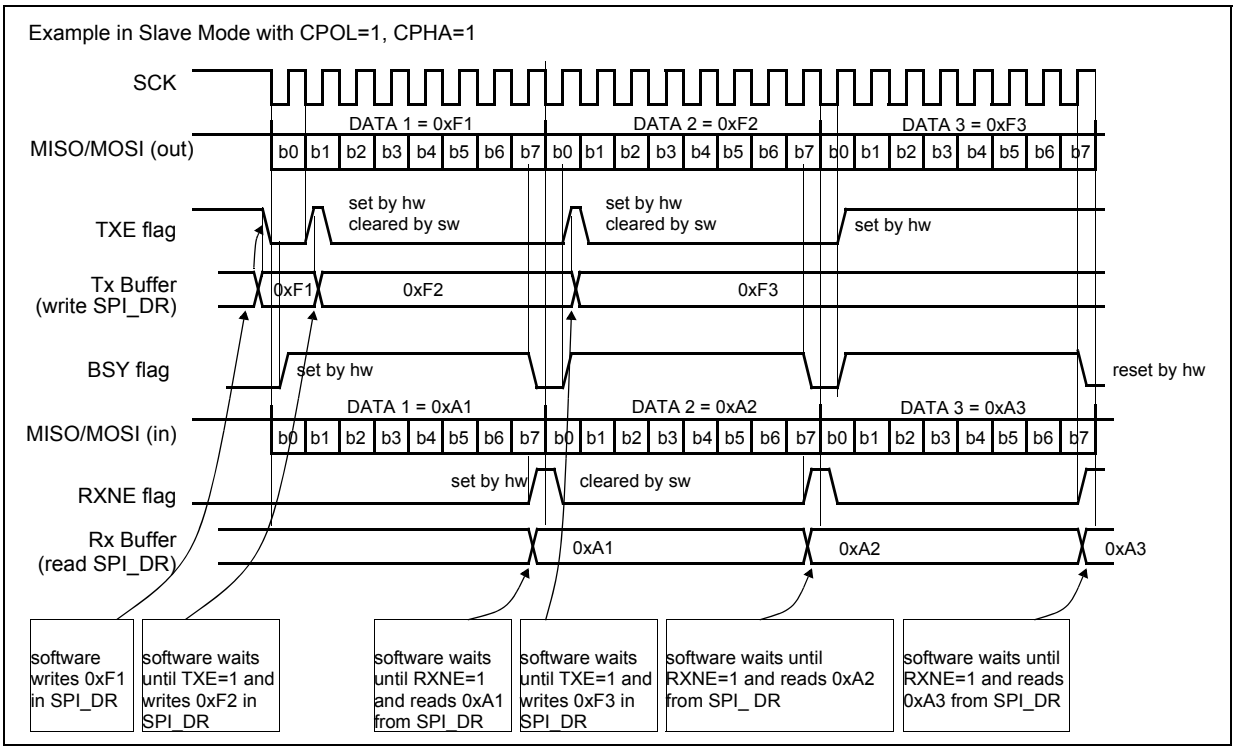


Figure 76. TXE/RXNE/BSY behavior in slave / full duplex mode (BDM = 0, RXONLY = 0). Case of continuous transfers



Transmit-only procedure (BDM = 0 RXONLY = 0)

In this mode, the procedure can be reduced as described below and the BSY bit can be used to wait until the effective completion of the transmission (see *Figure 75* and *Figure 76*):

1. Enable the SPI by setting the SPE bit
2. Write the first data to send in the SPI_DR register (this clears the TXE bit).
3. Wait until TXE = 1 and write the next data to be transmitted. Repeat this step for each data to be transmitted.
4. After writing the last data in the SPI_DR register, wait until TXE = 1 and then wait until BSY=0 which indicates that the transmission of the last data is complete.

This procedure can be also implemented using dedicated interrupt subroutines launched at each rising edge of TXE flag.

Note: 1 In master mode, during discontinuous communications, there is a 2 CPU clock period delay between the write operation to SPI_DR and the BSY bit setting. As a consequence, in transmit-only mode, it is mandatory to wait first until TXE is set and then until BSY is reset after having written the last data.

2 After transmitting two data in transmit-only mode, the OVR flag is set in the SPI_SR register since the received data are never read.

Figure 77. TXE/BSY in master transmit-only mode (BDM = 0 and RXONLY = 0). Case of continuous transfers

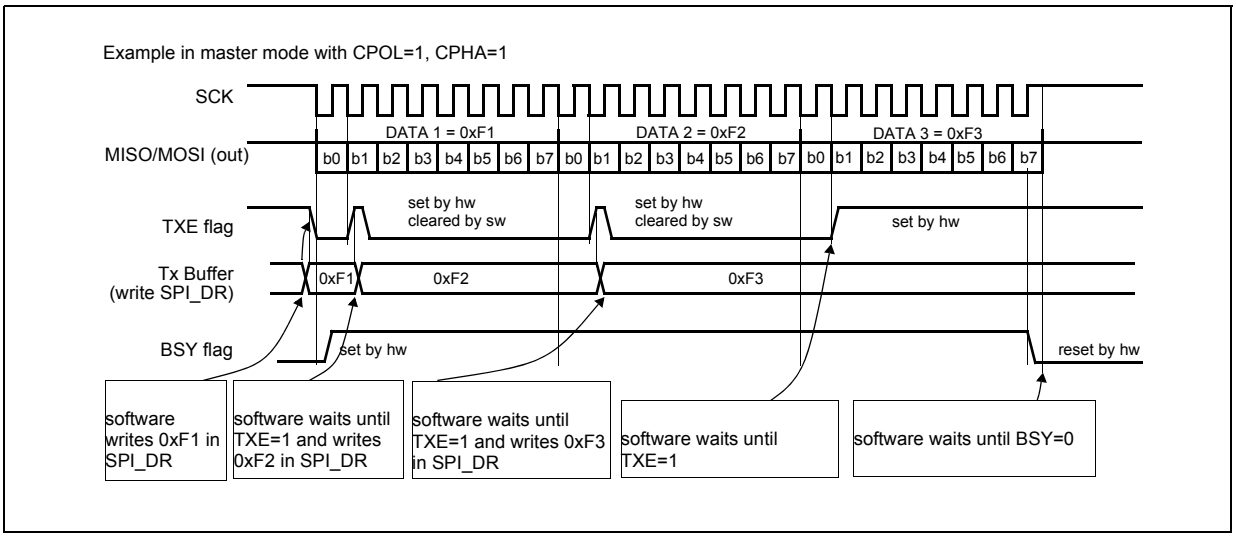
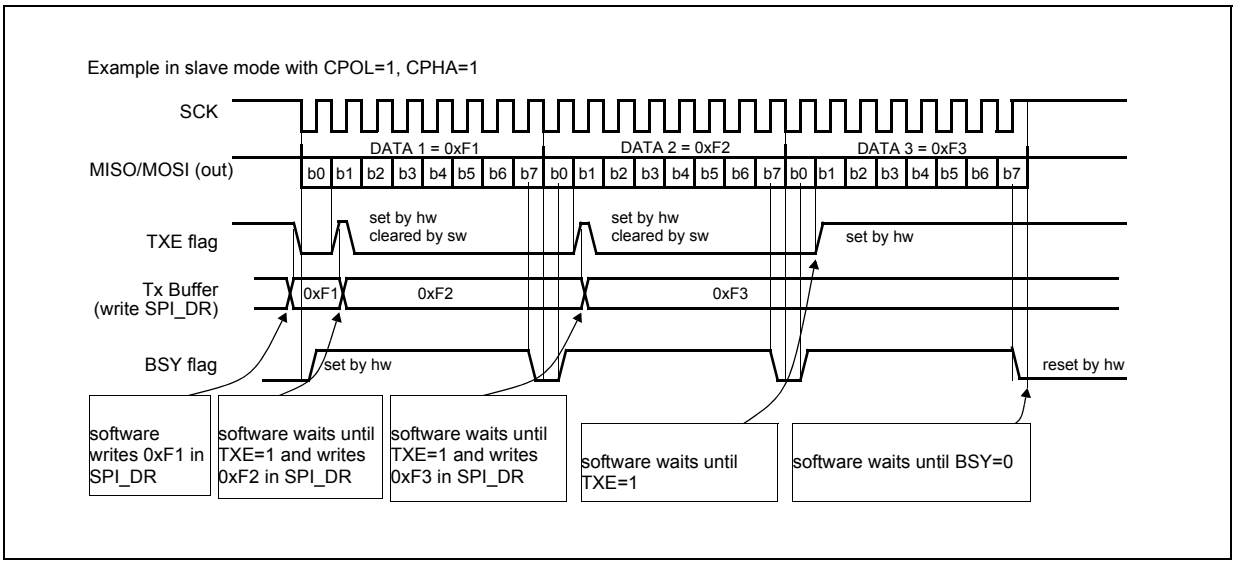


Figure 78. TXE/BSY in slave transmit-only mode (BDM = 0 and RXONLY = 0). Case of continuous transfers



Bidirectional transmit procedure (BDM = 1 and BDOE = 1)

In this mode, the procedure is similar to the Transmit-only procedure except that the BDM and BDOE bits must both be set in the SPI_CR2 register before enabling the SPI.

Unidirectional receive-only procedure (BDM = 0 and RXONLY = 1)

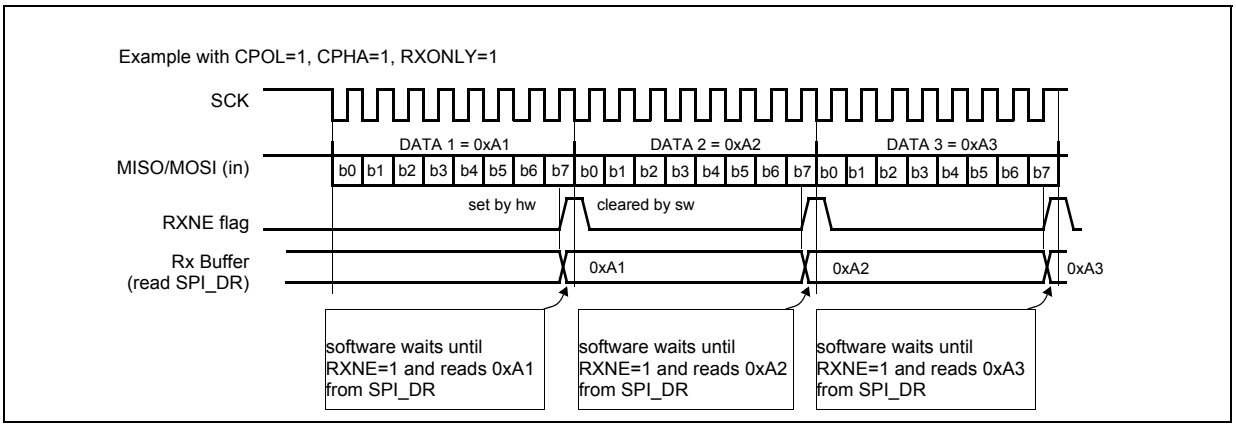
In this mode, the procedure can be reduced as described below (see *Figure 79*):

1. Set the RXONLY bit in the SPI_CR2 register
2. Enable the SPI by setting bit SPE to 1:
   a) In master mode, this immediately activates the generation of the SCK clock, and data is received serially until the SPI is disabled (SPE = 0).
   b) In slave mode, data are received when the SPI master device drives NSS low and generates the SCK clock.
3. Wait until RXNE =1 and read the SPI_DR register to get the received data (this clears the RXNE bit). Repeat this operation for each data to be received.

This procedure can be also implemented using dedicated interrupt subroutines launched at each rising edge of the RXNE flag.

Note: If it is required to disable the SPI after the last transfer, follow the recommendation described in Section 20.3.7: Disabling the SPI on page 215.

Figure 79. RXNE behavior in receive-only mode (BDM = 0 and RXONLY = 1). Case of continuous transfers



Bidirectional receive procedure (BDM = 1 and BDOE = 0)

In this mode, the procedure is similar to the Receive-only procedure except that the BDM bit must be set and the BDOE bit must be reset in the SPI_CR2 register before enabling the SPI.

Continuous and discontinuous transfers

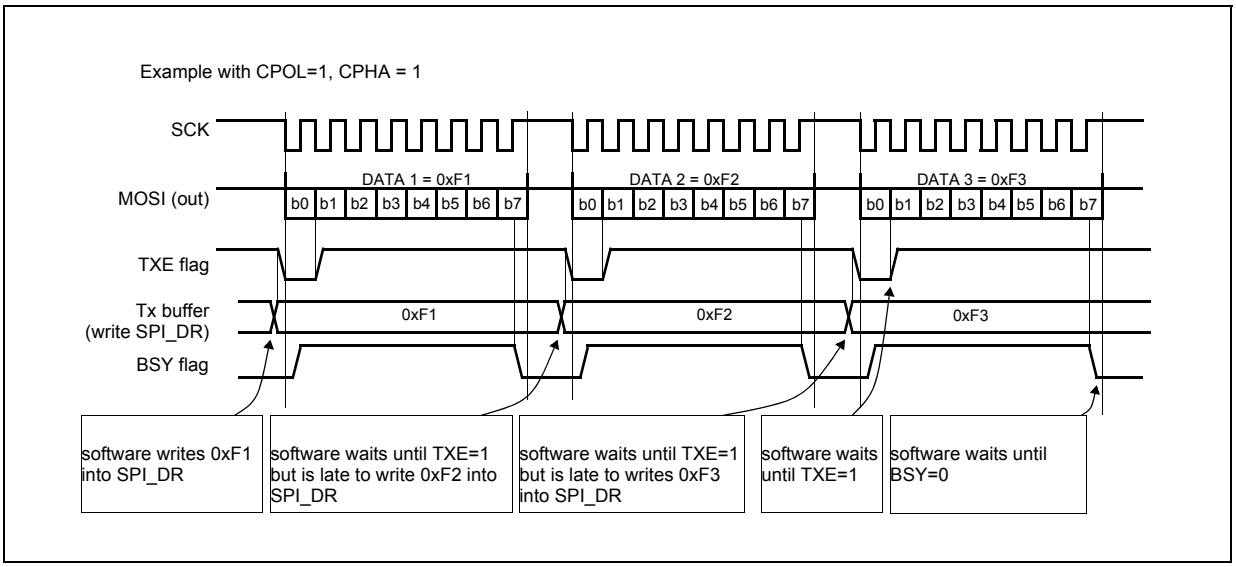
When transmitting data in master mode, if the software is fast enough to detect each TXE rising edge (or TXE interrupt) and to immediately write the SPI_DR register before the ongoing data transfer is complete, the communication is said to be continuous. In this case, there is no discontinuity in the generation of the SPI clock between each data and the BSY bit will never be reset between each data transfer.

On the contrary, if the software is not fast enough, this can lead to some discontinuities in the communication. In this case, the BSY bit is reset between each data transmission (see *Figure 80*).

In master receive-only mode (BDM = 0 and RXONLY = 1) or in bidirectional receive mode (BDM = 1 and BDOE = 0), the communication is always continuous and the BSY flag is always read at 1.

In slave mode, the continuity of the communication is decided by the SPI master device. But even if the communication is continuous, the BSY flag goes low between each transfer for a minimum duration of one SPI clock cycle (see *Figure 76*).

Figure 80. TXE/BSY behavior when transmitting (BDM = 0 and RXLONY = 0). Case of discontinuous transfers



20.3.6 Status flags

There are three status flags to allow the application to completely monitor the state of the SPI bus.

Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and that the next data to be transmitted can be loaded into the buffer. The TXE flag is reset when writing the SPI_DR register.

Rx buffer not empty (RXNE)

When set, this flag indicates that there is a valid received data in the Rx buffer. This flag is reset when SPI_DR is read.

Busy flag (BSY)

This BSY flag is set and reset by hardware (writing to this flag has no effect). The BSY flag indicates the state of the communication layer of the SPI.

When BSY is set, it indicates that the SPI is busy communicating. There is one exception in master mode / bidirectional receive mode (MSTR=1 and BDM=1 and BDOE=0) where the BSY flag is kept low during the reception.

The BSY flag is useful to detect the end of a transfer if the software wants to disable the SPI and enters Halt mode (or disable the peripheral clock). This will avoid corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is also useful to avoid write collisions in a multimaster system.

The BSY flag is set when a transfer starts with the exception of master mode / bidirectional receive mode (MSTR = 1 and BDM = 1 and BDOE = 0).

It is reset:

- when a transfer is finished (except in master mode if the communication is continuous)
- when the SPI is disabled
- when a master mode fault occurs (MODF = 1)

When communication is not continuous, the BSY flag is low between each communication.

When communication is continuous, in master mode, the BSY flag is kept high during the whole transfers.

When communication is continuous, in slave mode, the BSY flag goes back to low state for one SPI clock cycle between each transfer.

Note: Do not use the BSY flag to handle each data transmission or reception. It is better to use TXE and RXNE flags instead.

20.3.7 Disabling the SPI

When a transfer is terminated, the application can stop the communication by disabling the SPI peripheral. This is done by resetting the SPE bit.

For some configurations, disabling the SPI and entering Halt mode while a transfer is on-going, can cause the current transfer to be corrupted and/or it can happen that the BSY flag becomes unreliable.

To avoid any of these effects, it is recommended to respect the following procedure when disabling the SPI:

In master or slave full duplex mode (BDM = 0, RXONLY = 0):

1. Wait until RXNE = 1 to receive the last data
2. Wait until TXE = 1
3. Then wait until BSY = 0
4. Disable the SPI (SPE = 0) and eventually enter Halt mode (or disable the peripheral clock).

In master or slave unidirectional transmit-only mode (BDM = 0, RXONLY = 0) or bidirectional transmit mode (BDM = 1, BDOE = 1):

After the last data is written in the SPI_DR register:

1. Wait until TXE = 1
2. Then wait until BSY = 0
3. Disable the SPI (SPE = 0) and, if desired, enter Halt mode (or disable the peripheral clock).

In master unidirectional receive-only mode (MSTR = 1, BDM = 0, RXONLY = 1) or bidirectional receive mode (MSTR = 1, BDM = 1, BDOE = 0):

This case must be managed in a particular way to ensure that the SPI does not initiate a new transfer:

1. Wait for the second to last occurrence of RXNE = 1 (n-1)
2. Then wait for one SPI clock cycle (using a software loop) before disabling the SPI (SPE = 0)
3. Then wait for the last RXNE=1 before entering Halt mode (or disabling the peripheral clock).

Note: In master bidirectional receive mode (MSTR=1 and BDM=1 and BDOE=0), the BSY flag is kept low during a transfer.

In slave receive-only mode (MSTR = 0, BDM = 0, RXONLY = 1) or bidirectional receive mode (MSTR = 0, BDM = 1, BDOE = 0):

1. You can disable the SPI (write SPE = 1) whenever you want: the current transfer will complete before being effectively disabled.
2. Then, if you want to enter Halt mode, you must first wait until BSY = 0 before entering Halt mode (or disabling the peripheral clock).

20.3.8 Error flags

Master mode fault (MODF)

Master mode fault occurs when the master device has its NSS pin pulled low (in NSS hardware mode) or SSI bit low (in NSS software mode), this automatically sets the MODF bit. Master mode fault affects the SPI peripheral in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is reset. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is reset, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1. Make a read or write access to the SPI_SR register while the MODF bit is set.
2. Then write to the SPI_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state after this clearing sequence.

As a security, hardware does not allow you to set the SPE and MSTR bits while the MODF bit is set.

In a slave device the MODF bit cannot be set. However, in a multi-master configuration, the device can be in slave mode with this MODF bit set. In this case, the MODF bit indicates that there might have been a multimaster conflict for system control. You can use an interrupt routine to recover cleanly from this state by performing a reset or returning to a default state.

Overrun condition

An overrun condition occurs, when the master device has sent data bytes and the slave device has not cleared the RXNE bit resulting from the previous data byte transmitted. When an overrun condition occurs:

- OVR bit is set and an interrupt is generated if the ERRIE bit is set.

In this case, the receiver buffer contents will not be updated with the newly received data from the master device. A read to the SPI_DR register returns this byte. All other subsequently transmitted bytes are lost.

Clearing the OVR bit is done by a read access to the SPI_DR register followed by a read access to the SPI_SR register.

20.3.9 SPI low power modes

Table 42. SPI behavior in low power modes

Mode	Description
Wait	No effect on SPI. SPI interrupt events cause the device to exit from Wait mode.
Halt	SPI registers are frozen. In Halt mode, the SPI is inactive. If the SPI is in master mode, then communication resumes when the device is woken up by an interrupt with “wakeup from Halt mode” capability. If the SPI is in slave mode, then it can wake up the MCU from Halt mode after detecting the first sampling edge of data.

Using the SPI to wake up the device from Halt mode

When the microcontroller is in Halt mode, the SPI is still capable of responding as a slave provided the NSS pin is tied low or the SSI bit is reset before entering Halt mode.

When the first sampling edge of data (as defined by the CPHA bit) is detected:

- The WKUP bit is set in the SPI_SR register
- An interrupt is generated if the WKIE bit in the SPI_ICR register is set.
- This interrupt wakes up the device from Halt mode.
- Due to the time needed to restore the system clock, the SPI slave sends or receives a few data before being able to communicate correctly. It is then mandatory to use the following protocol:
  - A specific value is written into the SPI_DR before entering Halt mode. This value indicates to the external master that the SPI is in Halt mode
  - The external master sends the same byte continuously until it receives from the SPI slave device a new value other than the unique value indicating the SPI is in Halt mode. This new value indicates the SPI slave has woken-up and can correctly communicate.

Restrictions in receive-only modes

The wakeup functionality is not guaranteed in receive-only modes (BDM = 0 and RXONLY = 1 or BDM = 1 and BDOE = 0) since the time needed to restore the system clock can be greater than the data reception time. A loss of data in reception would then be induced and the slave device can not indicate to the master which data has been properly received.

20.3.10 SPI interrupts

Table 43. SPI interrupt requests

Interrupt event	Event flag	Enable control bit	Exit from Wait	Exit from Halt
Transmit buffer empty flag	TXE	TXIE	Yes	No
Receive buffer not empty flag	RXNE	RXIE	Yes	No
Wakeup event flag	WKUP	WKIE	Yes	Yes
Master mode fault event	MODF	ERRIE	Yes	No
Overrun error	OVR		Yes	No

20.4 SPI registers

20.4.1 SPI control register 1 (SPI_CR1)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
LSBFIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw			rw	rw	rw

Bit 7 **LSBFIRST**: Frame format (1)
0: MSB is transmitted first
1: LSB is transmitted first

Bit 6 **SPE**: SPI enable (2)
0: Peripheral disabled
1: Peripheral enabled

Bits 5:3 **BR[2:0]**: Baud rate control
000: $f_{MASTER}/2$
001: $f_{MASTER}/4$
010: $f_{MASTER}/8$
011: $f_{MASTER}/16$
100: $f_{MASTER}/32$
101: $f_{MASTER}/64$
110: $f_{MASTER}/128$
111: $f_{MASTER}/256$
Note: These bits should not be changed when the communication is ongoing.

Bit 2 **MSTR**: Master selection (1)
0: Slave configuration
1: Master configuration

Bit1 **CPOL**: Clock polarity (1)
0: SCK to 0 when idle
1: SCK to 1 when idle

Bit 0 **CPHA**: Clock phase (1)
0: The first clock transition is the first data capture edge
1: The second clock transition is the first data capture edge

1. This bit should not be changed when the communication is ongoing.
2. When disabling the SPI, follow the procedure described in *Section 20.3.7: Disabling the SPI on page 215*

20.4.2 SPI control register 2 (SPI_CR2)

Address offset: 0x01
Reset value: 0x00

7	6	5	4	3	2	1	0
BDM	BDOE	Reserved			RXOnly	SSM	SSI
rw	rw				rw	rw	rw

Bit 7 **BDM**: Bidirectional data mode enable
0: 2-line unidirectional data mode selected
1: 1-line bidirectional data mode selected

Bit 6 **BDOE**: Input/Output enable in bidirectional mode
This bit selects the direction of transfer in bidirectional mode when BDM is set to 1.
0: Input enabled (receive-only mode)
1: Output enabled (transmit-only mode)
In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.

Bits 5:3 Reserved

Bit 2 **RXONLY**: Receive only
0: Full duplex (Transmit and receive)
1: Output disabled (Receive only mode)
This bit combined with BDM bit selects the direction of transfer in 2 line uni-directional mode
This bit is also useful in a multi-slave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

Bit 1 **SSM**: Software slave management
0: Software slave management disabled
1: Software slave management enabled
When the SSM bit is set, the NSS pin input is replaced with the value coming from the SSI bit

Bit 0 **SSI**: Internal slave select
This bit has effect only when 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.
0: Slave mode
1: Master mode

20.4.3 SPI interrupt control register (SPI_ICR)

Address offset: 0x02
Reset value: 0x00

7	6	5	4	3	2	1	0
TXIE	RXIE	ERRIE	WKIE	Reserved			
rw	rw	rw	rw				

Bit 7 **TXIE**: Tx buffer empty interrupt enable
0: TXE interrupt masked
1: TXE interrupt not masked. This allows an interrupt request to be generated when the TXE flag is set.

Bit 6 **RXIE**: RX buffer not empty interrupt enable
0: RXNE interrupt masked
1: RXNE interrupt not masked. This allows an interrupt request to be generated when the RXNE flag is set.

Bit 5 **ERRIE**: Error interrupt enable
0: Error interrupt is masked
1: Error interrupt is enabled. This allows an interrupt request to be generated when an error condition occurs (OVR, MODF)

Bit 4 **WKIE**: Wakeup interrupt enable
0: Wakeup interrupt masked
1: Wakeup interrupt enabled. This allows an interrupt request to be generated when the WKUP flag is set.

Bits 3:0 Reserved

20.4.4 SPI status register (SPI_SR)

Address offset: 0x03
Reset value: 0x02

7	6	5	4	3	2	1	0
BSY	OVR	MODF	Reserved	WKUP	Reserved	TXE	RXNE
r	rc_w0	rc_w0		rc_w0		r	r

Bit 7 **BSY**: Busy flag
0: SPI not busy
1: SPI is busy in communication
This flag is set and reset by hardware.
Note: BSY flag must be used with cautious: refer to Section 20.3.6: Status flags on page 214 and Section 20.3.7: Disabling the SPI on page 215

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.

Bit 4 Reserved

Bit 3 **WKUP**: Wakeup flag
0: No wakeup event occurred
1: Wakeup event occurred
This flag is set on the first sampling edge on SCK when the STM8 is in Halt mode and the SPI is configured as slave.
This flag is reset by software writing 0.

Bit 2 Reserved

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

20.4.5 SPI data register (SPI_DR)

Address offset: 0x04
Reset value: 0x00

7	6	5	4	3	2	1	0
DR[7:0]							
rw							

Bits 7:0 **DR[7:0]**: Data register

Byte received or to be transmitted.

The data register is split into 2 buffers - one for writing (Transmit buffer) and another one for reading (Receive buffer). A write to the data register will write into the Tx buffer and a read from the data register will return the value held in the Rx buffer.

20.5 SPI register map and reset values

Table 44. SPI register map and reset values

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	SPI_CR1 Reset value	LSB FIRST 0	SPE 0	BR2 0	BR1 0	BR0 0	MSTR 0	CPOL 0	CPHA 0
0x01	SPI_CR2 Reset value	BDM 0	BDOE 0	- 000			RXONLY 0	SSM 0	SSI 0
0x02	SPI_ICR Reset value	TXIE 0	RXIE 0	ERRIE 0	WKIE 0	- 0000			
0x03	SPI_SR Reset value	BSY 0	OVR 0	MODF 0	- 0	WKUP 0	- 0	TXE 1	RXNE 0
0x04	SPI_DR Reset value	DR[7:0] 0							

21 Universal synchronous/asynchronous receiver transmitter (USART)

21.1 USART introduction

The USART (universal synchronous asynchronous receiver transmitter) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. It offers a very wide range of baud rates.

The USART supports synchronous one-way communication. The USART can be used for multiprocessor communication.

21.2 USART main features

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- High-precision baud rate generator system
  - Common programmable transmit and receive baud rates up to $f_{MASTER}/16$
- Programmable data word length (8 or 9 bits)
- Configurable STOP bits - support for 1 or 2 STOP bits
- Transmitter clock output for synchronous communication
- Separate enable bits for Transmitter and Receiver
- Transfer detection flags:
  - Receive buffer full
  - Transmit buffer empty
  - End of Transmission flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- 4 error detection flags:
  - Overrun error
  - Noise error
  - Frame error
  - Parity error
- 5 interrupt sources with flags:
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle line received
  - Parity error
- 2 interrupt vectors:
  - Transmitter interrupt
  - Receiver interrupt
- Reduced power consumption mode
- Multi-Processor communication - enter into mute mode if address match does not occur
- Wakeup from mute mode (by idle line detection or address mark detection)
- 2 receiver wakeup modes:
  - Address bit (MSB)
  - Idle line

21.3 USART functional description

The interface is externally connected to another device by three pins (see *Figure 81*). Any USART bidirectional communication requires a minimum of two pins: USART Receive data input (USART_RX) and USART transmit data output (USART_TX):

USART_RX is the serial data input. Over-sampling techniques are used for data recovery by discriminating between valid incoming data and noise.

USART_TX is the serial 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 pin is at high level.

Through these pins, serial data is transmitted and received in normal USART mode as frames including:

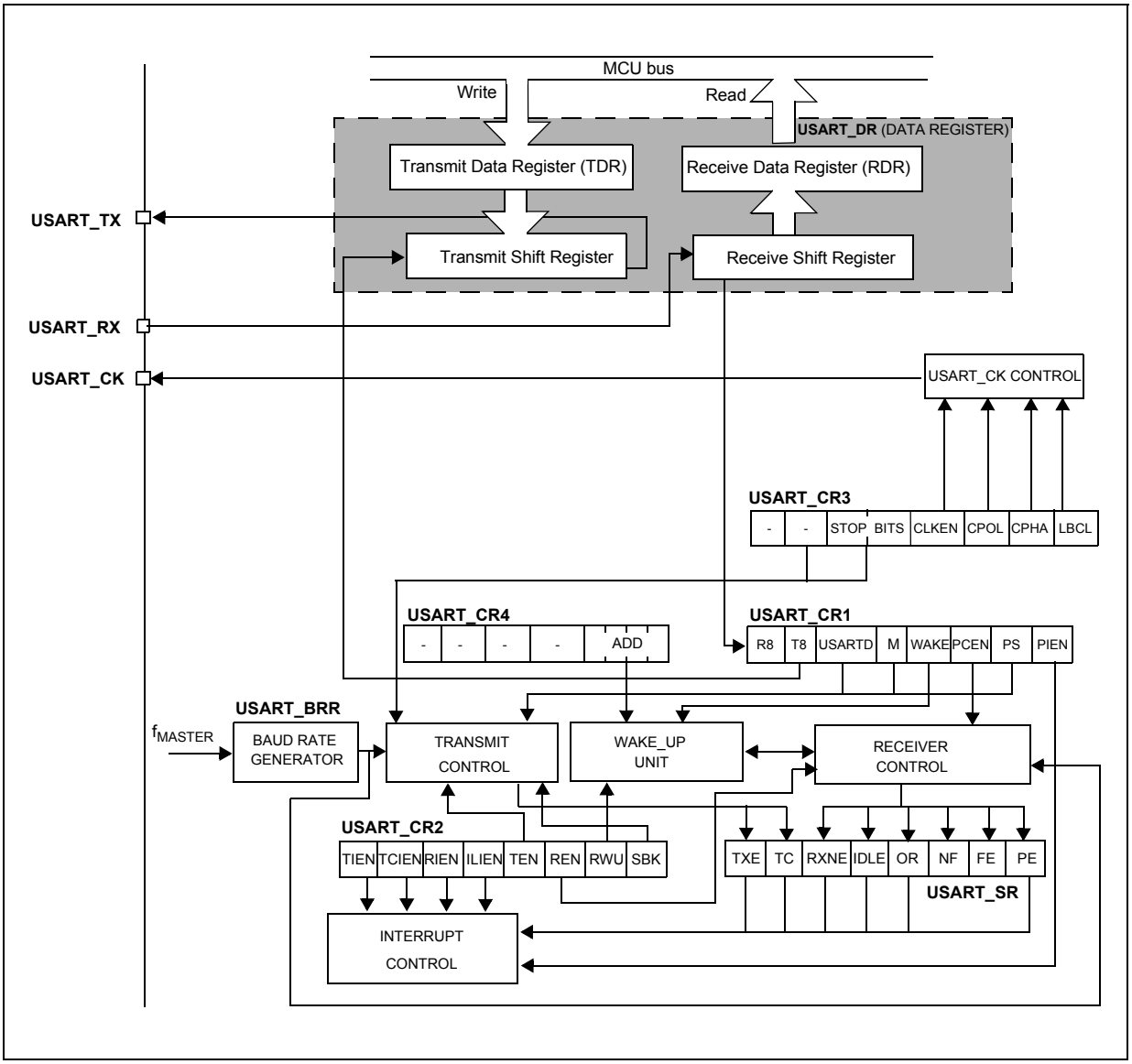
- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 1 and 2 STOP bits indicating that the frame is complete
- A status register (USART_SR)
- Data register (USART_DR)
- 16-bit baud rate prescaler (USART_BRR)

Refer to the register description for the definitions of each bit.

The following pin is required to interface in synchronous mode:

USART_CK: Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission (no clock pulses on start bit and STOP bit, and a software option to send a clock pulse on the last data bit). This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable.

Figure 81. USART block diagram



21.3.1 USART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the USART_CR1 register (see *Figure 82*).

The USART_TX pin is in low state during the start bit. It is in high state during the STOP bit.

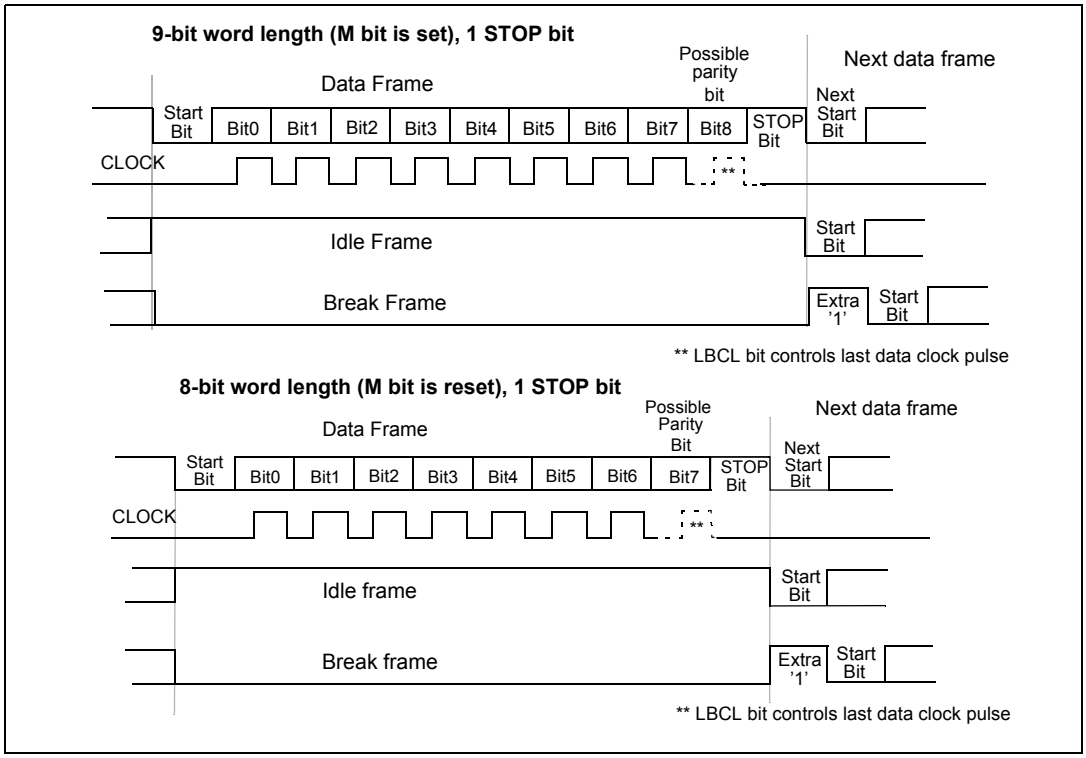
An ***Idle character*** is interpreted as an entire frame of “1”s (the number of “1” ‘s includes the start bit, the number of data bits and the number of STOP bits).

A ***Break character*** is interpreted on receiving “0”s for a frame period. At the end of the break frame the transmitter inserts either 1 or 2 STOP bits (logic “1” bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

Figure 82. Word length programming



21.3.2 Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the M bit is set, word length is 9 bits and the 9th bit (the MSB) has to be stored in the T8 bit in the USART_CR1 register.

When the transmit enable bit (TEN) is set, the data in the transmit shift register is output on the USART_TX pin and the corresponding clock pulses are output on the USART_CK pin.

Character transmission

During a USART transmission, data shifts out least significant bit first on the USART_TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see *Figure 81*).

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of STOP bits.

The following STOP bits are supported by USART.

Note: 1 The TEN bit should not be reset during transmission of data.Resetting the TEN bit during the transmission will corrupt the data on the USART_TX pin as the baud rate counters will get frozen.The current data being transmitted will be lost.

2 An idle frame will be sent after the TEN bit is enabled.

Configurable STOP bits during transmission

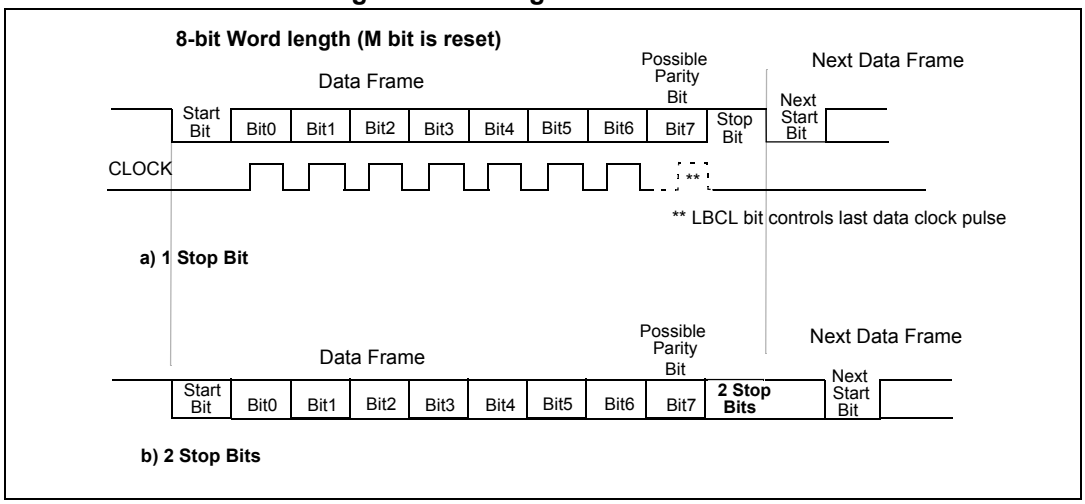
The number of STOP bits to be transmitted with every character can be programmed in Control register 3, bits 5,4.

- *1 STOP bit*: This is the default value of number of STOP bits.
- *2 STOP bits*: This will be supported by normal mode USART.

An idle frame transmission will include the STOP bits.

A break transmission consists of 10 low bits followed by the configured number of STOP bits (when m = 0) and 11 low bits followed by the configured number of STOP bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).

Figure 83. Configurable STOP bits



Procedure

1. Program the M bit in USART_CR1 to define the word length.
2. Program the number of STOP bits in USART_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
   a) USART_BRR2
   b) USART_BRR1
4. Set the TEN bit in USART_CR2 to enable transmitter mode.
5. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
6. Once the last data is written to the USART_DR register, wait until TC is set to ‘1’, which indicates that the last data transmission is complete. This last step is required, for instance, to avoid last data transmission corruption when disabling the USART or entering Halt mode.

Single byte communication

Clearing the TXE bit is always performed by a write to the data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

This flag generates an interrupt if the TIEN bit is set.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register. The data is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

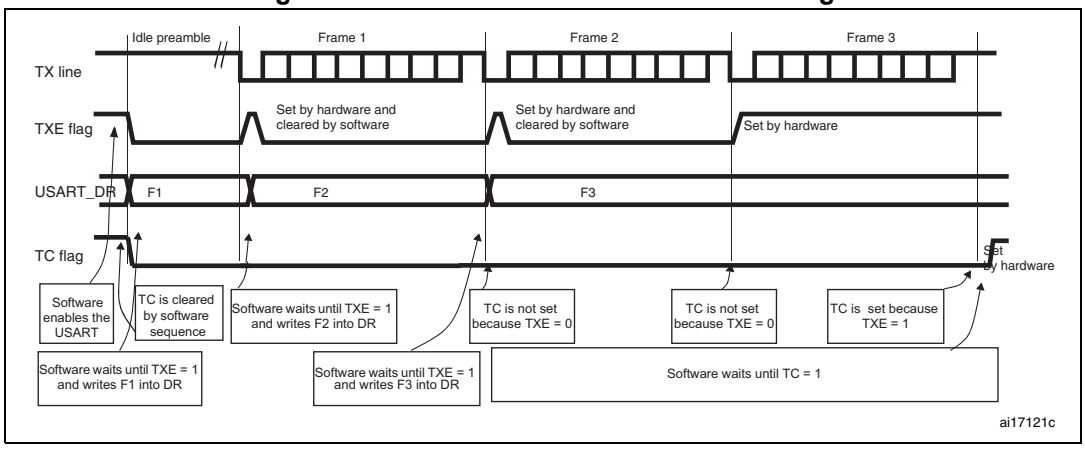
If a frame transmission is complete (after the stop bit) and the TXE bit is set, the TC bit is set. An interrupt is generated if the TCIEN is set in the USART_CR2 register. After writing the last data into the USART_DR register, it is mandatory to wait until TC is set to ‘1’ before entering Halt mode or disabling the USART (see *Figure 84: TC/TXE behavior when transmitting*).

Clearing the TC bit is performed by the following software sequence:

1. A read to the USART_SR register
2. A write to the USART_DR register

Note: The TC bit can also be cleared by writing a ‘0’ to it. This clearing sequence is recommended only for multibuffer communication.

Figure 84. TC/TXE behavior when transmitting



1. This example assumes that several other transmissions occurred after TE has been set. Otherwise an IDLE preamble would be transmitted first when writing to USART_DR for the first time.

Break character

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see *Figure 82*).

If the SBK bit is set to ‘1’ a break character is sent on the USART_TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the STOP bit of the break character).The USART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame.

Note: The break character is sent without taking into account the number of STOP bits. If the USART is programmed with 2 STOP bits, the Tx line is pulled low until the end of the first STOP bit only. Then 2 logic 1 bits are inserted before the next character.

Note: If the software resets the SBK bit before the start of break transmission, the break character is not transmitted. For two consecutive breaks, the SBK bit should be set after the STOP bit of the previous break.

Idle character

Setting the TEN bit drives the USART to send an idle frame before the first data frame.

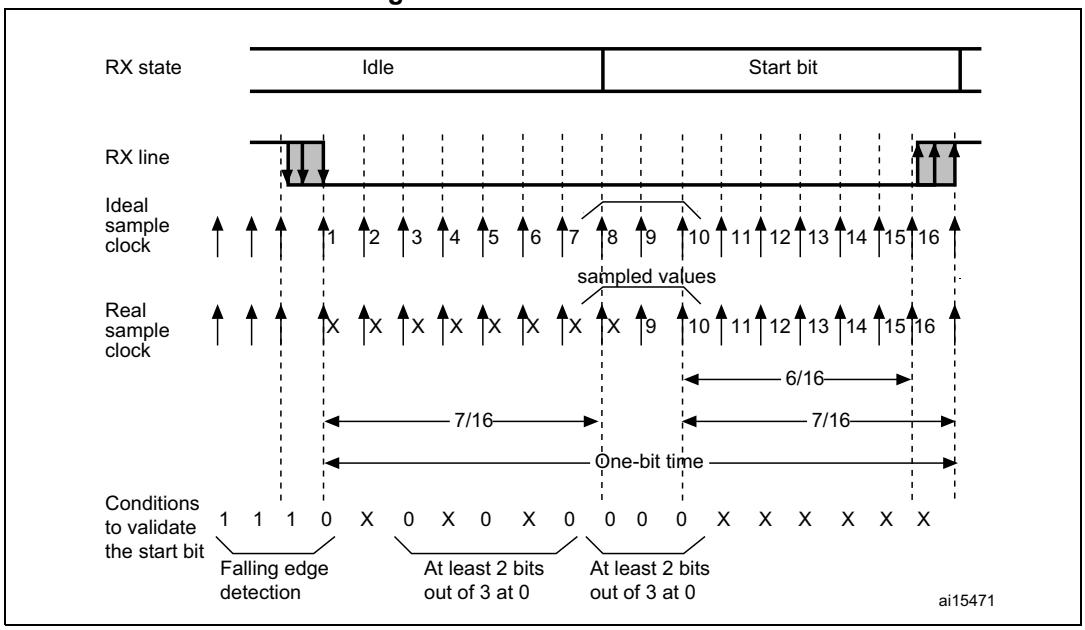
21.3.3 Receiver

The USART can receive data words of either 8 or 9 bits. When the M bit is set, word length is 9 bits and the MSB is stored in the R8 bit in the USART_CR1 register.

Start bit detection

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0X 0X 0 X 0X 0. The start bit detection sequence shown in *Figure 85*.

Figure 85. Start bit detection



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.*

If only 2 out of the 3 bits are at 0 (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), the start bit is validated but the NF noise flag bit is set.

The start bit is confirmed if the last 3 samples are at 0 (sampling on the 8th, 9th, and 10th bits.

Character reception

During a USART reception, data shifts in least significant bit first through the USART_RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register (see *Figure 2*).

Procedure:

1. Program the M bit in USART_CR1 to define the word length.
2. Program the number of STOP bits in USART_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
   a) USART_BRR2
   b) USART_BRR1
4. Set the REN bit USART_CR2. This enables the receiver which begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR.
- An interrupt is generated if the RIEN bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- Clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note: The REN bit should not be reset while receiving data. If the REN bit is disabled during reception, the reception of the current byte will be aborted.

Break character

When a break character is received, the USART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as a received data character plus an interrupt if the ILIEN 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.

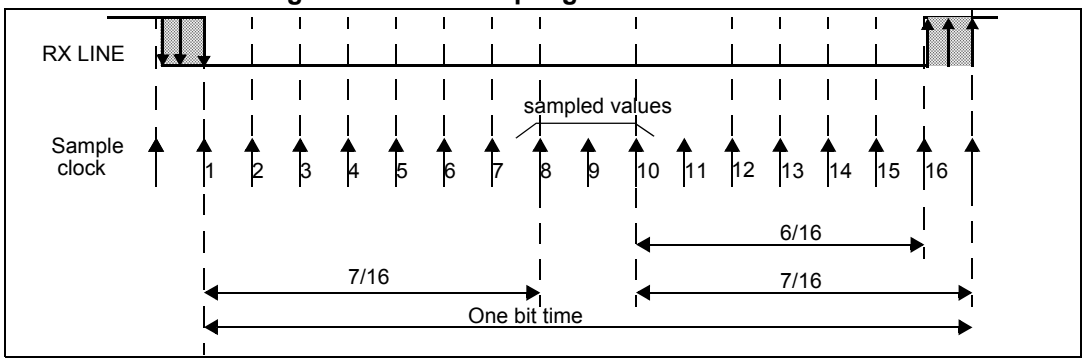
When an overrun error occurs:

- The OR bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_DR is performed.
- The shift register will be overwritten. The second data received during overrun is lost.
- An interrupt is generated if the RIEN bit is set.
- The OR bit is reset by a read to the USART_SR register followed by a USART_DR register read operation.

Noise error

Over-sampling techniques are used for data recovery by discriminating between valid incoming data and noise.

Figure 86. Data sampling for noise detection



Note: The sample clock frequency is 16x baud rate.

Table 45. Noise detection from sampled data

Sampled value	NF status	Received bit value	Data validity
000	0	0	Valid
001	1	0	Not Valid
010	1	0	Not Valid
011	1	1	Not Valid
100	1	0	Not Valid
101	1	1	Not Valid
110	1	1	Not Valid
111	0	1	Valid

When noise is detected in a frame:

- The NF is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART_DR register.

This bit rises at the same time as the RXNE bit which generates an interrupt. The NF bit is reset by a USART_SR register read operation followed by a USART_DR register read operation.

Framing error

A framing error is detected when:

The STOP bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART_DR register.
- No interrupt is generated in case of single byte communication. However, this bit rises at the same time as the RXNE bit which itself generates an interrupt.

The FE bit is reset by a USART_SR register read operation followed by a USART_DR register read operation.

Configurable STOP bits during reception

The number of STOP bits to be received can be configured through the control bits of Control Register 3 - it can be either 1 or 2.

1. *1 STOP bit*: Sampling for 1 STOP bit is done on the 8th, 9th and 10th samples.
2. *2 STOP bits*: Sampling for 2 STOP bits is done on the 8th, 9th and 10th samples of the first STOP bit.If a framing error is detected during the first STOP bit the framing error flag will be set. The second STOP bit is not checked for framing error. The RXNE flag will be set at the end of the first STOP bit.

21.3.4 High precision baud rate generator

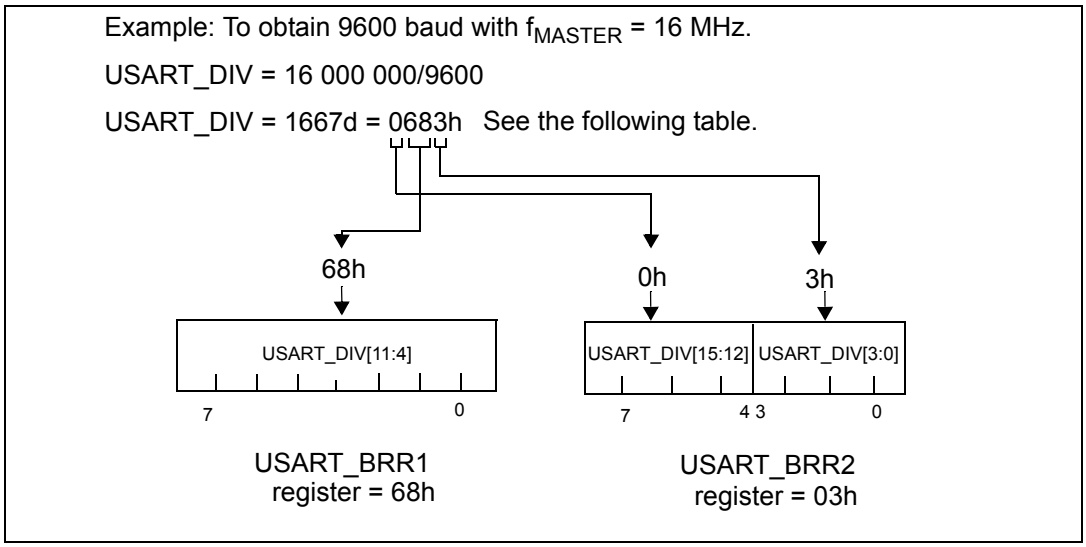
The receiver and transmitter (Rx and Tx) are both set to the same baud rate programmed by a 16-bit divider USART_DIV according to the following formula:

$$\text{Tx/ Rx baud rate} = \frac{f_{MASTER}}{USART_DIV}$$

The USART_DIV baud rate divider is an unsigned integer, coded in the BRR1 and BRR2 registers as shown in *Figure 87*.

Refer to *Table 46* for typical baud rate programming examples.

Figure 87. How to code USART_DIV in the BRR registers



Note: The Baud Counters will be updated with the new value of the Baud Registers after a write to BRR1. Hence the Baud Register value should not be changed during a transaction. The BRR2 should be programmed before BRR1.

Note: USART_DIV must be greater than or equal to 16d.

Table 46. Baud rate programming and error calculation

Baud rate	f_{MASTER} = 16 MHz				
In bps	Actual	% Error(1)	USART_DIV	BRR1	BRR2
2.4	2.4	0.0	2710h	71h	20h
9.6	9.6	0.0	09C4h	9Ch	04h
19.2	19.2	0.0	04E2	4Eh	02h
57.6	57.554	-0.08	01A1h	1Ah	01h
115.2	115.385	0.16	00D0h	0Dh	00h
230.4	230.769	0.16	0068h	06h	08h
460.8	461.538	0.16	0034h	03h	04h
921.6	923.077	0.16	001Ah	01h	0Ah

1. Error % = (Calculated - Desired) Baud Rate / Desired Baud Rate

Note: The lower the f_{MASTER} frequency, the lower the accuracy for a particular baud rate.The upper limit of the achievable baud rate can be fixed with the above data.

21.3.5 USART receiver's tolerance to clock deviation

The USART's asynchronous receiver works correctly only if the total clock system deviation is smaller than the USART receiver's tolerance. 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)

DTRA + DQUANT + DREC + DTCL < USART receiver's tolerance

The USART receiver's tolerance to properly receive data is equal to the maximum tolerated deviation and depends on the following choices:

- 10- or 11-bit character length defined by the M bit in the USART_CR1 register
- Use of fractional baud rate or not

Table 47. USART receiver's tolerance when USART_DIV[3:0] is 0

M bit	NF is an error	NF is don't care
0	3.75%	4.375%
1	3.41%	3.97%

Table 48. USART receiver's tolerance when USART_DIV[3:0] is different from 0

M bit	NF is an error	NF is don't care
0	3.33%	3.88%
1	3.03%	3.53%

Note: The figures specified in Table 47 and Table 48 may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit times when M=0 (11-bit times when M=1).

21.3.6 Parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCEN 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 49*.

Table 49. Frame formats

M bit	PCEN bit	USART frame
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7-bit data | PB | STB |
1	0	| SB | 9-bit data | STB |
1	1	| SB | 8-bit data PB | STB |

Legends: SB: Start Bit, STB: STOP bit, PB: Parity bit

Note: In case of wakeup by an address mark, the MSB bit of the data is taken into account and not the parity bit

Even parity: the parity bit is calculated to obtain an even number of "1s" inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Ex: data=00110101; 4 bits set => parity bit will be 0 if even parity is selected (PS bit in USART_CR1 = 0).

Odd parity: the parity bit is calculated to obtain an odd number of "1s" inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Example: data=00110101; 4 bits set => parity bit will be 1 if odd parity is selected (PS bit in USART_CR1 = 1).

Transmission: If the PCEN bit is set in USART_CR1 then the MSB bit of the data written in the data register is not transmitted but is changed by the parity bit to give an even number of '1's if even parity is selected (PS=0) or an odd number of '1's if odd parity is selected (PS=1).

Reception: If the parity check fails, the PE flag is set in the USART_SR register and an interrupt is generated if the PIEN bit is set in the USART_CR1 register.

21.3.7 Multi-processor communication

It is possible to perform multiprocessor communication with the USART (several USARTs connected in a network). For example, one of the USARTs can be the master, its Tx output is connected to the Rx input of the other USART. The others are slaves, their respective Tx outputs are logically ANDed together and connected to the Rx input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART_CR2 register is set to 1. RWU can be controlled automatically by hardware or written by the software under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART_CR1 register:

- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

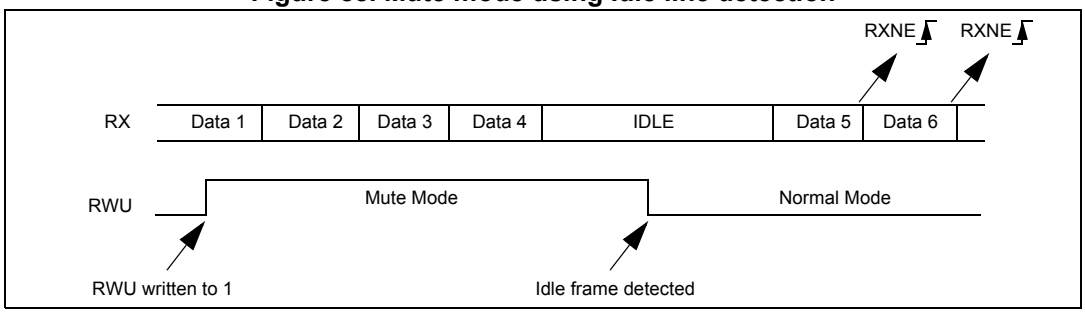
Idle line detection (WAKE=0)

The USART enters mute mode when the RWU bit is written to 1.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART_SR register. RWU can also be written to 0 by software.

An example of mute mode behavior using idle line detection is given in *Figure 88*.

Figure 88. Mute mode using Idle line detection



Address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a ‘1’ else they are considered as data. In an address byte, the address of the targeted receiver is put on the 4 LSB. This 4-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART_CR4 register.

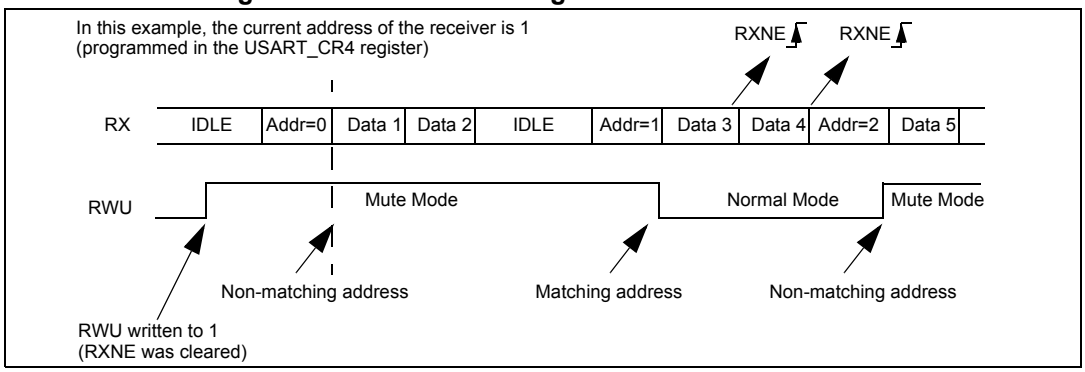
The USART enters mute mode when an address character is received which does not match its programmed address. The RXNE flag is not set for this address byte and no interrupt request is issued as the USART would have entered mute mode.

It exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

The RWU bit can be written to 0 or 1 when the receiver buffer contains no data (RXNE=0 in the USART_SR register). Otherwise the write attempt is ignored.

An example of mute mode behavior using address mark detection is given in *Figure 89*.

Figure 89. Mute mode using address mark detection



Note: If parity control is enabled, the parity bit remains in the MSB and the address bit is put in the "MSB - 1" bit.

For example, with 7-bit data, address mode and parity control:

SB I 7-bit data I ADD I PB I STB

where:

SB = Start Bit
STB = STOP Bit
ADD = Address bit
PB = Parity Bit

21.3.8 USART synchronous communication

The USART transmitter allows the user to control bidirectional synchronous serial communications in master mode.

Note: This feature is only available for devices with USART_CK pin. Check the device pinout for availability.

The USART_CK pin is the output of the USART transmitter clock. No clock pulses are sent to the USART_CK pin during start bit and STOP bit. Depending on the state of the LBCL bit in the USART_CR3 register clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the USART_CR3 register allows the user to select the clock polarity, and the CPHA bit in the USART_CR3 register allows the user to select the phase of the external clock (see *Figure 90*, *Figure 91* & *Figure 92*). During idle and break frames, the external CK clock is not activated.

In synchronous mode, the USART receiver works differently compared to asynchronous mode. If REN=1, the data is sampled on USART_CK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time (even if the hold time is not relevant due to the SPI protocol) must be respected (which depends on the baud rate: 1/16 bit time for an integer baud rate).

During the idle state, preamble phase and break transmission, the external USART_CK clock is not activated. In synchronous mode, the USART transmitter works exactly like in asynchronous mode. But as USART_CK is synchronized with USART_TX (depending on CPOL and CPHA), the data on USART_TX is synchronous. In this mode, the USART receiver works slightly differently compared to the asynchronous mode: if REN=1, the data is still sampled using the internal oversampling clock and the baud rate clock is output on the USART_CK pin (rising or falling edge is aligned with the data sampling event depending on CPOL and CPHA). But contrary to asynchronous mode, the data is evaluated using one sample and not the majority of 3 samples, meaning that the NF bit will never be set.

Setup and hold times must be respected (depending on the baud rate: 1/16 bit time for an integer baud rate).

Note: 1 The USART_CK pin works in conjunction with the USART_TX pin. When the USART transmitter is disabled (TEN and REN= 0), the USART_CK and USART_TX pins go into high impedance state.

2 The LBCL, CPOL and CPHA bits in USART_CR3 have to be selected when both the transmitter and the receiver are disabled (TEN=REN=0) to ensure that the clock pulses function correctly. These bits should not be changed while the transmitter or the receiver is enabled.

3 It is recommended to set TEN and REN are set in the same instruction in order to minimize the setup and the hold time of the receiver.

4 The USART supports master mode only: it cannot receive or send data related to an input clock (USART_CK is always an output).

5 The data given in this section apply only when the USART_DIV[3:0] bits in the USART_BRR2 register are kept at 0. Else the setup and hold times are not 1/16 of a bit time but 4/16 of a bit time.

This option allows to serially control peripherals which consist of shift registers, without losing any functions of the asynchronous communication which can still talk to other asynchronous transmitters and receivers.

Figure 90. USART example of synchronous transmission

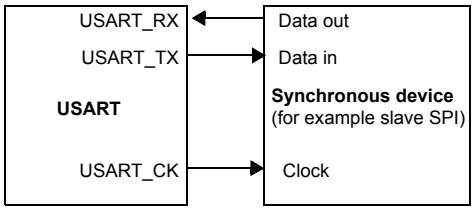


Figure 91. USART data clock timing diagram (M=0)

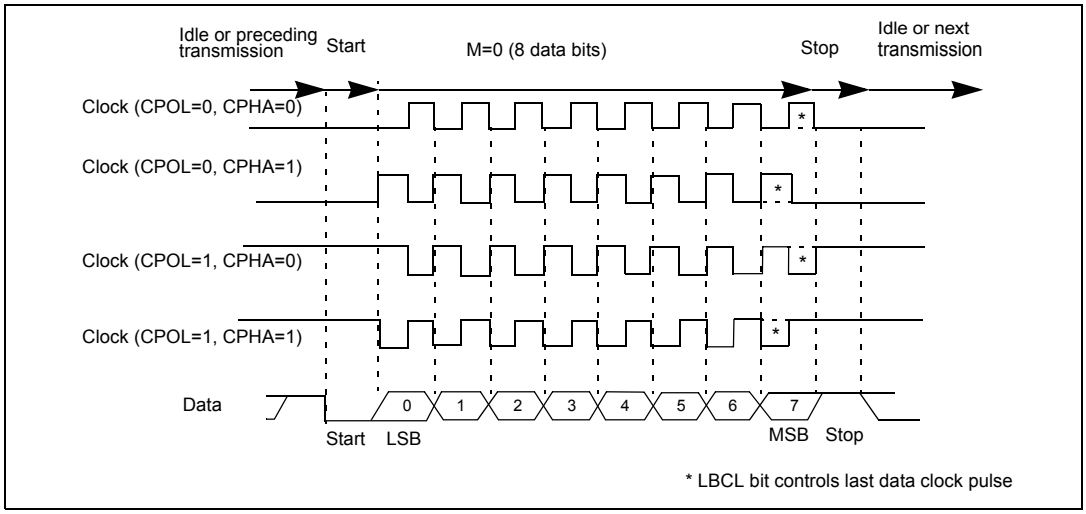


Figure 92. USART data clock timing diagram (M=1)

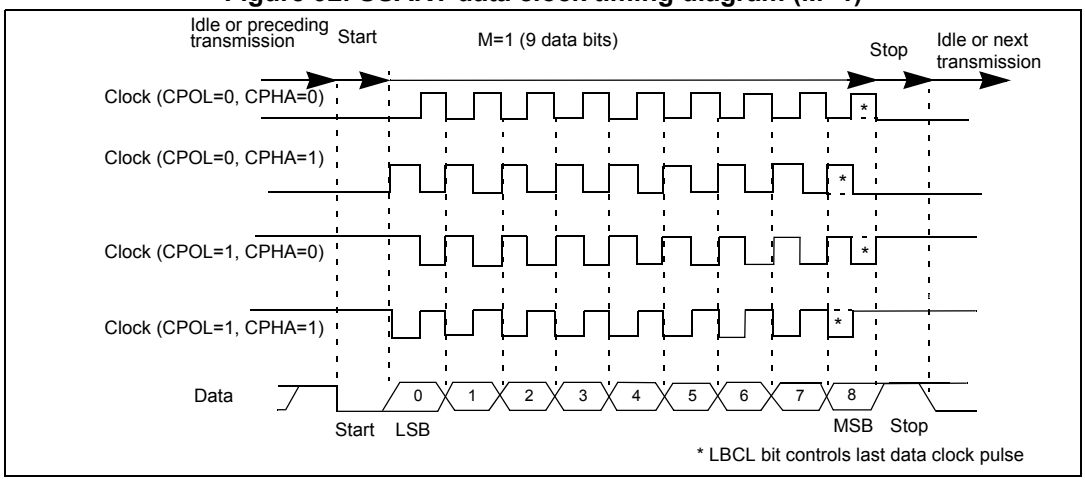
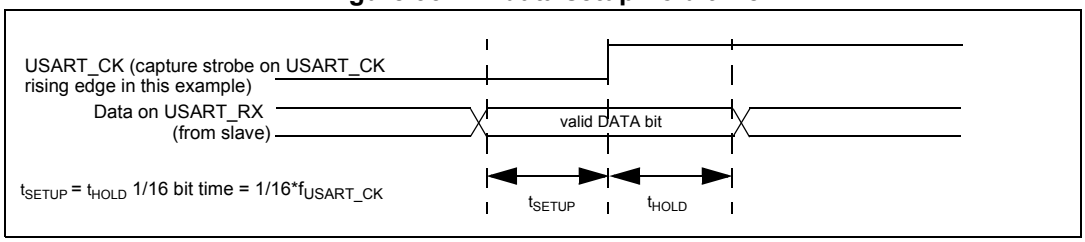


Figure 93. RX data setup/hold time



21.4 USART low power modes

Table 50. USART interface behavior in low power modes

Mode	Description
Wait	No effect on USART. USART interrupts cause the device to exit from Wait mode.
Halt	USART registers are frozen. In Halt mode, the USART stops transmitting/receiving until Halt mode is exited.

21.5 USART interrupts

Table 51. USART interrupt requests

Interrupt event	Event flag	Enable control bit	Exit from Wait	Exit from Halt
Transmit data register empty	TXE	TIEN	Yes	No
Transmission complete	TC	TCIEN	Yes	No
Received data ready to be read	RXNE	RIEN	Yes	No
Overrun error detected	OR		Yes	No
Idle line detected	IDLE	ILIEN	Yes	No
Parity error	PE	PIEN	Yes	No

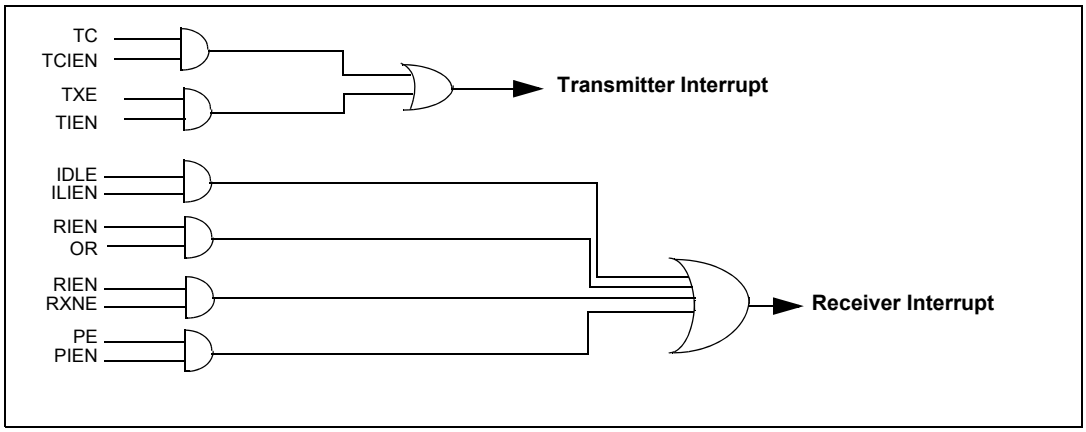
Note: 1 The USART interrupt events are connected to two interrupt vectors (see Figure 94).

a) Transmission Complete or Transmit Data Register empty interrupt.

b) Idle line detection, Overrun error, Receive data register full, Parity error interrupt, and Noise flag.

2 These events generate an interrupt if the corresponding enable control bit is set and the interrupt mask in the CCR register is reset (RIM instruction).

Figure 94. USART interrupt mapping diagram



21.6 USART registers

21.6.1 Status register (USART_SR)

Address offset: 0x00
Reset value: 0xC0

7	6	5	4	3	2	1	0
TXE	TC	RXNE	IDLE	OR	NF	FE	PE
r	rc_w0	r	r	r	r	r	r

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TIEN bit =1 in the USART_CR2 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register
1: Data is transferred to the shift register

Bit 6 **TC**: Transmission complete

TC bit is set by hardware if the transmission of a frame containing data is complete and TXE bit is set. An interrupt is generated if TCIEN=1 in the USART_CR2 register.

TC bit is cleared either by a software sequence (a read to the USART_SR register followed by a write to the USART_DR register), or by programming the bit to ‘0’. This clear sequence is recommended only for multibuffer communications.

0: Transmission is not complete
1: Transmission is complete

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_DR register. An interrupt is generated if RIEN=1 in the USART_CR2 register. It is cleared by a read to the USART_DR register.

0: Data is not received
1: Received data is ready to be read.

Bit 4 **IDLE**: IDLE line detected (1)

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the ILIEN=1 in the USART_CR2 register. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No Idle Line is detected
1: Idle Line is detected

Bit 3 **OR**: Overrun error(2)

This bit is set by hardware when the word currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. An interrupt is generated if RIEN=1 in the USART_CR2 register. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No Overrun error
1: Overrun error is detected

Bit 2 **NF**: Noise flag (3)

This bit is set by hardware when noise is detected on a received frame. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No noise is detected
1: Noise is detected

Bit 1 **FE**: Framing error [4]

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by a software sequence (a read to the USART_SR register followed by a read to the USART_DR register).

0: No framing error is detected

1: Framing error or break character is detected

Bit 0 **PE**: Parity error

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by a software sequence (a read to the status register followed by a read to the USART_DR data register). You have to wait for the RXNE flag to be set before clearing it. An interrupt is generated if PIEN=1 in the USART_CR1 register.

0: No parity error

1: Parity error

1. The IDLE bit is not set again until the RXNE bit has been set itself (i.e. a new idle line occurs)
2. When this bit is set, the RDR register content is not lost but, the shift register is overwritten.
3. This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt.
4. This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. If the word currently being transferred causes both a frame error and an overrun error, it is transferred and only the OR bit is set.

21.6.2 Data register (USART_DR)

Address offset: 0x01

Reset value: 0xXX

7	6	5	4	3	2	1	0
DR[7:0]							
rw							

Bits 7:0 **DR[7:0]**: Data value

Contains the received or transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register.

The RDR register provides the parallel interface between the input shift register and the internal bus.

21.6.3 Baud rate register 1 (USART_BRR1)

The baud rate registers are common to both the transmitter and the receiver. The baud rate is programmed using two registers BRR1 and BRR2. Writing of BRR2 (if required) should precede BRR1, since a write to BRR1 will update the baud counters.

See *Figure 87: How to code USART_DIV in the BRR registers on page 235* and *Table 46: Baud rate programming and error calculation on page 236*.

Note: The baud counters stop counting if the TEN or REN bits are disabled respectively.

Address offset: 0x02
Reset value: 0x00

7	6	5	4	3	2	1	0
USART_DIV[11:4]							
rw	rw	rw	rw	-	rw	rw	rw

Bits 7:0 **USART_DIV[11:4]**: USART_DIV bits (1)
These 8 bits define the 2nd and 3rd nibbles of the 16-bit USART divider (USART_DIV).

1. BRR1 = 0x00 means USART clock is disabled.

21.6.4 Baud rate register 2 (USART_BRR2)

Address offset: 0x03

Reset value: 0x00

7	6	5	4	3	2	1	0
USART_DIV[15:12]				USART_DIV[3:0]			
rw				rw			

Bits 7:4 **USART_DIV[15:12]**: MSB of USART_DIV
These 4 bits define the MSB of the USART Divider (USART_DIV)

Bits 3:0 **USART_DIV[3:0]**: LSB of USART_DIV
These 4 bits define the LSB of the USART Divider (USART_DIV)

21.6.5 Control register 1 (USART_CR1)

Address offset: 0x04
Reset value: 0x00

7	6	5	4	3	2	1	0
R8	T8	USARTD	M	WAKE	PCEN	PS	PIEN
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **R8**: Receive data bit 8
This bit is used to store the 9th bit of the received word when M=1

Bit 6 **T8**: Transmit data bit 8
This bit is used to store the 9th bit of the transmitted word when M=1

Bit 5 **USARTD**: USART disable (for low power consumption)
When this bit is set the USART prescaler and outputs are stopped at the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.
0: USART enabled
1: USART prescaler and outputs disabled

Bit 4 **M**: word length
This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 Data bits, ‘n’ STOP bit (n depending on STOP[1:0] bits in the USART_CR3 register)
1: 1 Start bit, 9 Data bits, 1 STOP bit
Note: The M bit must not be modified during a data transfer (both transmission and reception)

Bit 3 **WAKE**: Wakeup method
This bit determines the USART wakeup method, it is set or cleared by software.
0: Idle line
1: Address mark

Bit 2 **PCEN**: 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, PCEN is active after the current byte (in reception and in transmission).
0: Parity control disabled
1: Parity control enabled

Bit 1 **PS**: Parity selection
This bit selects the odd or even parity when the parity generation/detection is enabled (PCEN bit set). It is set and cleared by software. The parity will be selected after the current byte.
0: Even parity
1: Odd parity

Bit 0 **PIEN**: Parity interrupt enable
This bit is set and cleared by software.
0: Parity interrupt disabled
1: Parity interrupt is generated whenever PE=1 in the USART_SR register

21.6.6 Control register 2 (USART_CR2)

Address offset: 0x05
Reset value: 0x00

7	6	5	4	3	2	1	0
TIEN	TCIEN	RIEN	ILIEN	TEN	REN	RWU	SBK
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **TIEN**: Transmitter interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever TXE=1 in the USART_SR register

Bit 6 **TCIEN**: Transmission complete interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever TC=1 in the USART_SR register

Bit 5 **RIEN**: Receiver interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever OR=1 or RXNE=1 in the USART_SR register

Bit 4 **ILIEN**: IDLE Line interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An USART interrupt is generated whenever IDLE=1 in the USART_SR register

Bit 3 **TEN**: Transmitter enable (1) (2)
This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled

Bit 2 **REN**: Receiver enable
This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit

Bit 1 **RWU**: Receiver wakeup(3) (4)
This bit determines if the USART is in mute mode or not. It is set and cleared by software and can be cleared by hardware when a wakeup sequence is recognized.
0: Receiver in active mode
1: Receiver in mute mode

Bit 0 **SBK**: Send break
This bit set is used to send break characters. It can be set and cleared by software. It should be set by software, and will be reset by hardware during the STOP bit of break.
0: No break character is transmitted
1: Break character will be transmitted

1. During transmission, a “0” pulse on the TEN bit (“0” followed by “1”) sends a preamble (idle line) after the current word.
2. When TEN is set there is a 1 bit-time delay before the transmission starts.
3. Before selecting Mute mode (by setting the RWU bit) the USART must first receive a data byte, otherwise it cannot function in Mute mode with wakeup by Idle line detection.
4. In address mark detection wakeup configuration (WAKE bit=1) the RWU bit cannot be modified by software while the RXNE bit is set.

21.6.7 Control register 3 (USART_CR3)

Address offset: 0x06

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		STOP[1:0]		CLKEN	CPOL	CPHA	LBCL
		rw		rw	rw	rw	rw

Bit 7:6 Reserved

Bits 5:4 **STOP**: STOP bits

These bits are used for programming the STOP bits.

00: 1 STOP bit
01: Reserved
10: 2 STOP bits
11: Reserved

Bit 3 **CLKEN**: Clock enable

This bit allows the user to enable the USART_CK pin.

0: USART_CK pin disabled
1: USART_CK pin enabled

Bit 2 **CPOL**: Clock polarity(1)

This bit allows the user to select the polarity of the clock output on the USART_CK pin. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: USART_CK to 0 when idle
1: USART_CK to 1 when idle.

Bit 1 **CPHA**: Clock phase (1)

This bit allows the user to select the phase of the clock output on the USART_CK pin. It works in conjunction with the CPOL bit to produce the desired clock/data relationship

0: The first clock transition is the first data capture edge
1: The second clock transition is the first data capture edge

Bit 0 **LBCL**: Last bit clock pulse(1)(2)

This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the USART_CK pin.

0: The clock pulse of the last data bit is not output to the USART_CK pin.
1: The clock pulse of the last data bit is output to the USART_CK pin.

1. These 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.
2. The last bit is the 8th or 9th data bit transmitted depending on the 8 or 9 bit format selected by the M bit in the USART_CR1 register.

21.6.8 Control register 4 (USART_CR4)

Address offset: 0x07

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				ADD[3:0]			
				rw			

Bit 7:4 Reserved

Bits 3:0 **ADD[3:0]**: Address of the USART node

This bit-field gives the address of the USART node.

This is used in multiprocessor communication during mute mode, for wakeup with address mark detection.

21.6.9 USART register map and reset values

Table 52. USART register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	USART_SR Reset value	TXE 1	TC 1	RXNE 0	IDLE 0	OR 0	NF 0	FE 0	PE 0
0x01	USART_DR Reset value	DR[7:0] xxxxxxxx							
0x02	USART_BRR1 Reset value	USART_DIV[11:4] 00000000							
0x03	USART_BRR2 Reset value	USART_DIV[15:12] 0000				USART_DIV[3:0] 0000			
0x04	USART_CR1 Reset value	R8 0	T8 0	USARTD 0	M 0	WAKE 0	PCEN 0	PS 0	PIEN 0
0x05	USART_CR2 Reset value	TIEN 0	TCIEN 0	RIEN 0	ILIEN 0	TEN 0	REN 0	RWU 0	SBK 0
0x06	USART_CR3 Reset value	- 00		STOP 00		CKEN 0	CPOL 0	CPHA 0	LBCL 0
0x07	USART_CR4 Reset value	- 0000				ADD[3:0] 0000			

22 ProxSense™ (PXS)

22.1 Introduction

The ProxSense peripheral measures a capacitance variation so that applications can detect the proximity or touch of one or more conductive objects when the electrodes are connected to the PXS pins.

This peripheral operates in **Projected mode** which is used to measure the charge transferred by a driven electrode to a second electrode.

Note: ProxSense ™ is a trademark of Azoteq (Pty) Ltd.

22.2 Main features of the ProxSense peripheral

- 10 independent receiver channels, allowing 10 measurements to be performed in parallel.
- Each of the 10 receiver channels can be associated with two different pins, effectively allowing an application to have up to 20 receiver channels.
- Each receiver channel can be independently configured to perform projected capacitance measurements.
- The size of each C_S capacitor can be independently configured with 5 bits of resolution.
- The size of each electrode parasitic capacitance compensation (EPCC) capacitor can be independently configured with 8 bits of resolution.
- The launch of each conversion can be synchronized with an external input, allowing for very precise measurements in the presence of cyclic disturbances (such as 50 Hz/60 Hz power-supply circuits).
- End of conversion interrupt, which has an associated interrupt flag, and can wake the CPU from Active-halt mode.
- First channel completion interrupt, which has an associated interrupt flag, and can wake the CPU from Active-halt mode after the first channel has finished its measurement.
- The frequency of the clock driving the charge transfer protocol (based on HSI_PXS) can be scaled from 125 kHz up to 16 MHz.
- The duration of the UP and PASS phases can be configured independently to optimize acquisition frequency.
- The noise detection feature, when enabled, indicates if there was noise during a previous conversion.
- Response time can be enhanced by specifying a maximum conversion measurement value, which is mainly useful for touchkey applications.

22.3 PXS pins

Table 53. PXS pins

Name	Signal type	Description
PXS_RX*n*A/ PXS_RX*n*B	Analog input/output	20 receivers, each associated with one of the 10 internal C_S capacitors into which the electrode charge is transferred (RXA*n* and RXB*n* share one Cs).
PXS_TX*m*	Digital output	15 transmit outputs, each of which can drive an electrode that transfers a charge (through a projected capacitor) into a second electrode connected to a Rx pin.
PXS_RFIN	Analog input	Antenna input to detect radio frequency (RF) noise.
PXS_TRIG	Digital input	External trigger for the acquisition.
PXS_VREG	External decoupling capacitor	A capacitance must be connected to stabilize the dedicated ProxSense regulator.

Note: m = 0 to 14 (transmit outputs) and n = 0 to 9 (receivers).

22.4 Functional description

22.4.1 General description

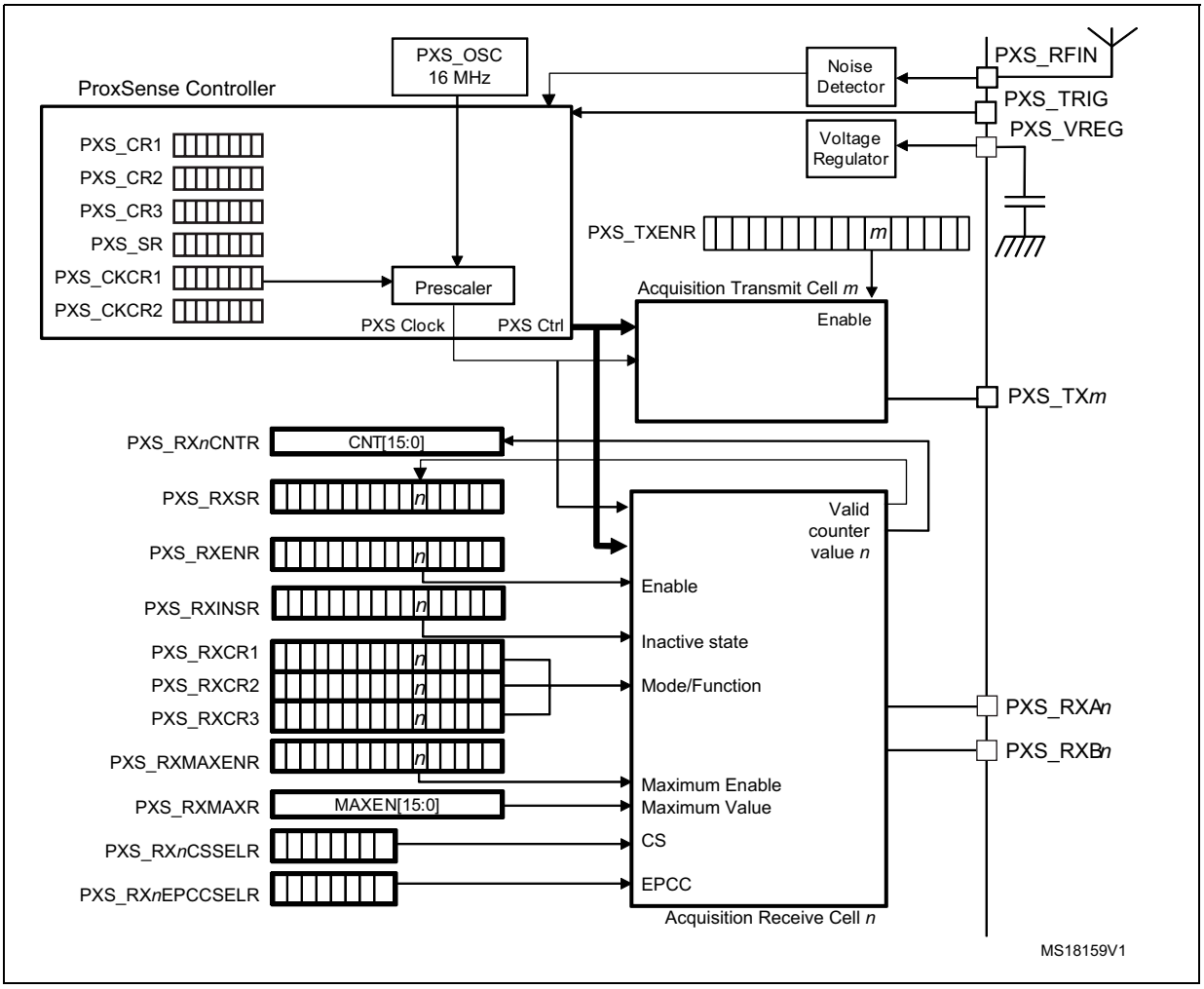
Figure 95 shows the block diagram of the ProxSense peripheral.

There are 10 receiver channels so that the capacitance on 10 electrodes can be measured in parallel. Two pins are associated with each receiver channel; one is from the A group and the other is from the B group, enabling the measurement of 20 electrodes.

Each receiver channel converts the capacitance it measures on an electrode into a 16-bit value (PXS_RX*n*CNTR). It acquires this result by repeatedly transferring a charge from the electrode to an internal capacitor, Cs. The resulting value is the number of charge transfers which are necessary to raise the voltage on the Cs above a reference voltage. Thus, a larger capacitance results in a smaller value.

In Projected Capacitance mode (selected using PXS_RXCR*n*), the capacitance between a Tx electrode and the corresponding Rx electrode is measured. In this case, the acquisition gives larger data values as a conductive object approaches the intersection of these two electrodes. This is because the Tx electrode starts to couple more with the conductive object and less with the Rx electrode.

Figure 95. ProxSense block diagram



There are 15 transmit outputs (m = 0 to 14) and up to 10 receivers (n = 0 to 9), each cell having 2 PXS_RX pins (A and B).

22.4.2 ProxSense charge transfer sequence

A ProxSense acquisition consists of a series of charge transfers, each of which is divided into an UP phase and a PASS phase. The acquisition result is the number of charge transfers which are necessary to cause the internal capacitor (Cs), which receives each of the charge transfers, to reach the threshold voltage.

The events which occur during the UP and PASS phases are defined in *Table 54*.

Table 54. Activities during the UP and PASS phases

Mode	UP phase	PASS phase
Projected capacitance	– Tx line forced low – Rx line held at sampled voltage (sampled at end of previous PASS phase).	– Tx line forced high – Cs absorbs the charge transferred to the Rx line through the projected capacitance.

Configuring the charge transfer sequence

The ProxSense clock can be prescaled (using PRESC[2:0] in the PXS_CKCR1 register) which serves as the basis for generating the UP and PASS phases.

The lengths of the two phases can be independently configured using UPLEN[2:0] and PASSLEN[2:0] (PXS_CKCR2). To allow additional flexibility, setting the bit INCPHASE (PXS_CKCR1) causes each of the UP and PASS phase lengths to be increased by one half of a ProxSense clock cycle.

The length of the non-overlap period (the deadtime between the UP and PASS phases) is determined by ANADEAD (PXS_CKCR1). When ANADEAD = ‘0’, there is a one-half cycle deadtime between the UP and PASS phases. When ANADEAD = ‘1’, the deadtime is basically zero cycles with each of the UP and PASS cycles shorted by about 10 ns to ensure non-overlapping. Setting ANADEAD to ‘1’ increases the UP phase by one full cycle.

In the reset configuration with bits ANADEAD and INCPHASE set to ‘0’ and bits UPLEN and PASSLEN set to ‘1’, a charge transfer cycle lasts four t_{PXS}. For instance, when PRESC[2:0] = ‘111’, f_{PXS} = 16 MHz and each charge transfer cycle lasts 250 ns.

22.5 ProxSense operations

22.5.1 PXS on/off control

The ProxSense is enabled and disabled by respectively setting and resetting the PXSEN bit in the PXS_CR1 register. Before enabling the ProxSense, certain PXS registers must be initialized as they are only writable when PXSEN = 0.

22.5.2 PXS initialization

Before writing in any of the PXS registers, the corresponding peripheral clock must be enabled in the CLK_PCKENR1 register.

Initialization before enabling the ProxSense

After enabling the corresponding peripheral clock, the clock frequency in the PXS_CKCR1 register and the length of the UP and PASS phases in PXS_CKCR2 must be set. It is also recommended at this stage to select ANADEAD and INCPHASE in PXS_CKCR1.

The stabilization time, the conversion completion threshold, and the sample and hold strength selection must be selected in PXS_CR3.

If the noise detection is used it must be enabled at this stage in the PXS_CR2 register.

Acquisition initialization

These initializations can be made when the PXS is enabled in PXS_CR1, but not when a conversion is ongoing.

- Enable optionally the interrupts, the synchronization feature, and the Rx coupling in PXS_CR2.
- Select the mode for the receivers in PXS_RXCR1/2/3. In a standard application, this is done only once for all acquisitions.
- Select the state of the pin when the Rx is disabled (inactive) in the PXS_RXINSR register. The state of the disabled PXS_TX*m* pins is configured by the corresponding GPIO. For instance, to have PXS_TX0 grounded when inactive, PD0 must be configured as output low level (PD_DR = 0bxxxxxxx0). To configure TX0 as high impedance, PD0 must be configured as output open-drain high level (PD_DR = 0bxxxxxxx1).
- Set the maximum count limit in PXS_MAXR and enable it for the selected receivers in PXS_MAXENR.
- Select the sampling capacitor (Cs) size in PXS_RX*n*CSSELR registers. It is recommended to select the same Cs size for all receivers belonging to the same object, especially when a position is interpolated from the measurements of some receivers (for example, sliders, wheels or screens).
- Select the receiver group channels A or B in the PXS_CR2 register
- Select the receivers and the transmitters by setting their corresponding bits respectively in PXS_RXENR and PXS_TXENR.
- Select the electrode parasitic capacitance compensation size according to the acquisition configuration in PXS_RX*n*EPCCSELR registers. This size has to be adapted for each couple (Rx, Tx).

The last three actions should be repeated before each acquisition with a different transmitter.

22.5.3 PXS acquisition launching

Manual start

The acquisition is launched on all enabled receivers in parallel by setting the START bit in PXS_CR1.

Synchronized start

To synchronize an acquisition with an external signal connected to the PXS_TRIG pin, the SYNC feature must be enabled and the signal edge selected in PXS_CR2. The acquisition is enabled by setting the START bit in PXS_CR1 and the acquisition starts on the next occurrence of the selected edge on the PXS_TRIG pin. Once a synchronized acquisition is completed, the START bit must be set again to start a new acquisition.

The start cannot be synchronized during Active-halt mode as transitions on the PXS_TRIG input are not recognized during Active-halt mode.

22.5.4 PXS acquisition result

The application can check that the acquisition is fully completed by polling the EOCF bit in PXS_ISR. Alternatively, the application can poll the FCCF bit in PXS_ISR to find out when at least one channel has finished its acquisition. Otherwise, instead of polling EOCF or FCCF, the corresponding interrupts can be enabled. If the PXS_RXSR and PXS_RXENR registers are identical, all the selected Rx have been successfully acquired. Each channel result is available in PXS_RX*n*CNTR.

The status flags EOCF and FCCF in PXS_ISR are cleared by setting the START bit in PXS_CR1 again.

If the noise detection is enabled, its flag must be checked before processing the data.

22.5.5 Stopping a PXS acquisition

The only way to stop a conversion which is in progress is by writing ‘0’ to PXSEN. In this case, the results of the conversion which was stopped prematurely are not guaranteed.

Caution: CIPF can stay at ‘1’ for as long as 8 µs after PXSEN is cleared.

22.6 Special features

22.6.1 Sampling capacitor selection

As the sampling capacitors are built-in, their size can be adapted to the connected electrode. 32 sizes of sampling capacitances (C_S) can be selected for each receiver using the PXS_RX*n*CSSELR register.

22.6.2 EPCC selection

The electrode parasitic capacitance compensation (EPCC) first fine-tunes the C_S capacitance value so that all receiver conversion results are close to the same targeted value. Second, it can be used to compensate for changes in the environment conditions. This compensation makes the system more sensitive to changes and thus smaller capacitance changes on the electrode can be detected.

The EPCC size is adjusted in the PXS_RX*n*EPCCSELR register.

22.6.3 RF detection

Besides on-chip protection which prevents RF contamination, the product provides a feature to detect RF interference.

An antenna has to be connected to the PXS_RFIN pin, and the NOISEDETEN bit must be set in the PXS_CR2 register before enabling the ProxSense by setting PXS_EN bit in PXS_CR1. Then, after each acquisition, the application can check the NOISEDETF bit in the PXS_ISR register to verify if the results should be discarded. The NOISEDETF bit must be reset by the application.

22.7 Low power modes

22.7.1 Behavior in low power modes

Table 55. Behavior in low power modes

Mode	Description
Wait	PXS interface is active. Interrupt events cause the device to exit from Wait and Active-halt.
Active-halt	
Halt	PXS interface is not active.

22.7.2 Reducing power consumption

As for other peripherals, the consumption can be reduced when the peripheral is not used by disabling the ProxSense and switching its clock in the CLK_PCKENiR register.

LOW_POWER bit

Setting the LOW_POWER bit in PXS_CR1 powers on the analog part of the ProxSense only while a conversion is active. This introduces a stabilization time just after the START bit is set, so in case of successive conversions with time constraints, it is recommended to clear the LOW_POWER bit just before setting the START bit to avoid this stabilization between the acquisitions.

Stabilization time

The ProxSense peripheral manages its stabilization automatically. In other words, if software or the SYNC feature requests that a conversion is launched before it is stabilized, the conversion is deferred and started automatically as soon as stabilization is complete.

The length of time required for the analog circuits to stabilize after powering on the ProxSense is inversely related to the amount of current sunk by the ProxSense circuitry during a conversion.

Examples include:

- An extra 420 µs of stabilization time might be required if only a single channel is active, with a charge-transfer frequency of 4 MHz, with BIAS = ‘10’, and with 10 pF on the Tx line.
- An extra 120 µs of stabilization time is required if several configured channels are active.

The duration of the stabilization time can be reduced using the STAB[1:0] bits in register PXS_CR3. This can reduce the overall current consumption of the application. By default, a period of 1.7 ms is selected.

Other parameters to reduce consumption

The principle factor of power consumption in the ProxSense module is the conversion time. For optimal power consumption, it is necessary to select the lowest acquisition length that will not adversely affect sensitivity.

To further reduce power consumption, it is recommended to ground the Rx pins when they are disabled by resetting the PXS_RXINSR register and to configure the GPIO pins connected to Tx channels as output low level when they are not driven by the ProxSense.

If the Rx lines do not interact with each other, disabling Rx coupling by resetting the RXCOUPLING bit in PXS_CR2, will also reduce power consumption as the Rx line drive is stopped as soon as the corresponding C_S reaches the threshold voltage (V_{THR}).

The sample and hold strength is confgured using the BIAS[1:0] bits in PXS_CR3 to set the minimum value that will allow the final discharged voltage to be maintained while the transmit lines (Tx) are transitioning back to low.

22.8 PXS interrupts

The ProxSense has two interrupt request sources:

- End of conversion (EOC) interrupt, which is activated when all of the enabled channels have completed their acquisitions.
- First channel conversion completion (FCC) interrupt, which is activated as soon as one channel completes its acquisition.

To enable interrupt requests for each interrupt source, set the required EOCITEN or FCCITEN interrupt enable bit in PXS_CR2.

Before enabling an interrupt, it is recommended to clear the EOCF and FCCF status flags in PXS_ISR to avoid triggering an interrupt due to a previously completed acquisition.

Before exiting the interrupt routine, the EOCF and FCCF status flags must be cleared by writing ‘0’ in the corresponding bit in PXS_ISR.

Note: These interrupts enable the device to exit from Wait and Active-halt modes but, not from Halt mode.

22.9 ProxSense registers

22.9.1 ProxSense Control Register 1 (PXS_CR1)

Address offset: 0x00
Reset value: 0x00

7	6	5	4	3	2	1	0
PXSEN	START	LOW_POWER	Reserved				
rw	r0_w	rw					

Bit 7 **PXSEN**: ProxSense enable

0: ProxSense feature is disabled and ProxSense analog circuits are powered down

1: ProxSense feature is activated

This bit is reset to 0 only by software.

Note: If PXSEN is set to 0 (by software) during a conversion, the conversion is stopped and the conversion results are not guaranteed.

Bit 6 **START**: Start conversion

0: Writing zero has no effect

1: Start conversion

Writing 1 to this bit starts a new conversion if the ProxSense peripheral is enabled (if PXSEN = 1) and if a conversion is not already in progress (if CIPF = 0). START is always read as ‘0’.

Note: Writing 1 when a conversion is already in progress (when CIPF = 1) has no effect.

Note: Writing 1 when the ProxSense peripheral is disabled (when PXSEN = 0) has no effect unless PXSEN is being set with the same write instruction.

Bit 5 **LOW_POWER**: Low power mode

0: ProxSense analog is powered on whenever PXSEN = 1

1: ProxSense analog is powered on only during conversions (while CIPF = 1 and SYNCPF = 0)

Bits 4:0 Reserved

22.9.2 ProxSense Control Register 2 (PXS_CR2)

Address offset: 0x01
Reset value: 0x00

7	6	5	4	3	2	1	0
EOCITEN	FCCITEN	NOISEDETEN	Reserved	RXGROUP	RXCOUPLING	SYNCEN	SYNCEDGE
rw	rw	rw		rw	rw	rw	rw

Bit 7 **EOCITEN**: End of conversion interrupt enable

0: End of conversion interrupt disabled

1: End of conversion interrupt enabled

When enabled, an interrupt occurs when EOCF = 1, i.e., once all the conversions on the receivers have been completed. EOCF must be cleared during the interrupt service routine.

Bit 6 **FCCITEN**: First conversion completion interrupt enable

0: First conversion completion interrupt disabled

1: First conversion completion interrupt enabled

When enabled, an interrupt occurs when FCCF = 1, i.e., a first conversion is completed on one of the enabled receivers. Subsequent conversion completions on other channels do not trigger an interrupt, even if FCCF was previously cleared. Software can determine which channels have completed their conversions by reading the PXS_RXSR register.

Bit 5 **NOISEDETEN**: Noise detection enable

0: RF Noise detection disabled

1: RF Noise detection enabled

When NOISEDETEN = 1, the RF noise detection circuit is enabled. An antenna should be connected to the PXS_RFIN pin to detect RF noise properly. NOISEDETF in PXS_ISR indicates if noise was detected during a conversion.

Note: This bit can be modified only when PXSEN = 0.

Bit 4 Reserved

Bit 3 **RXGROUP**: Rx group selection

0: Rx ‘A’ group (RX0A to RX9A) is active

1: Rx ‘B’ group (RX0B to RX9B) is active

Note: When RXGROUP = 0, the Rx ‘B’ group is in the inactive state (as specified by PXS_RXINSR), and vice-versa.

Note: This bit can be modified only when CIPF = 0.

Bit 2 **RXCOUPLING**: Reduce coupling between receiver lines

0: The charge transfer cycle stops on a Rx line when its corresponding Cs has reached the threshold voltage (V_{THR}).

1: Each Rx which is activated (RXEN*n* = 1) continues to toggle until all of the activated Rx channels have reached the threshold voltage (V_{THR}) or a timeout occurs.

For applications where the Rx lines have significant coupling between each other, it may be beneficial to set this bit so that the parasitic coupling on a given Rx line does not increase after a neighboring Rx channel has reached the threshold voltage.

Resetting this bit to 0 can reduce power consumption since some Rx lines would likely toggle for shorter periods of time.

Note: This bit can be modified only when CIPF = 0.

Bit 1**SYNCEN**: Enable synchronization (SYNC) feature

0: Disable the SYNC feature

1: Enable the SYNC feature

This feature cannot be used with Active-halt mode: transitions on the PXS_TRIG input are not recognized during Active-halt mode.

Bit 0**SYNCEDGE**: Synchronization edge selection

0: Launch conversion on the falling edge of the PXS_TRIG input (when SYNCEN = 1)

1: Launch conversion on the rising edge of the PXS_TRIG input (when SYNCEN = 1)

22.9.3 ProxSense Control Register 3 (PXS_CR3)

Address offset: 0x02
Reset value: 0x04

7	6	5	4	3	2	1	0
STAB[1:0]		BIAS[1:0]		VTHR[3:0]			
rw		rw		rw			

Bits 7:6 **STAB[1:0]**: Selection for stabilization time after ProxSense power-on

00: 1.7 ms, can be used for all configurations.

01: 500 µs, can be used when at least one channel with moderate consumption (>170 µA) is activated.

10: 120 µs, can be used when several channels are activated in Projected mode

11: Reserved, do not use this selection

This stabilization time is applied at the beginning of the first conversion after each time the ProxSense is powered on (on every conversion when LOW_POWER = 1).

Bits 5:4 **BIAS[1:0]**: Sample and hold strength selection

00: 2.5 µA

01: 5 µA

10: 7.5 µA

11: 10 µA

This is the nominal bias current for the OpAmps which maintain the PXS_RX*n* pins at the final (discharged) voltage while the transmit lines are transitioning (back) low. This function is valid only for projected capacitance measurements. A higher bias is needed when the projected capacitance (the capacitance between each Rx line and each Tx line) is high.

Note: This register can be modified only when PXSEN = 0.

Bits 3:0 **VTHR[3:0]**: Threshold voltage (V_{THR}) selection

0000: 0.5 x VREG

...

0100: 0.61 x VREG (default after reset)

...

1111: 0.9 x VREG

The threshold voltage can be set to values between 0.5 * V_{REG} and 0.9 * V_{REG} using VTHR[3:0]:

$V_{REG} = 0.02667 \times V_{THR} \times V_{REG} + V_{REG}/2$

When the voltage on a given sampling capacitor (C_S) reaches V_{THR}, the conversion for the corresponding receive channel is completed.

Note: This register can be modified only when PXSEN = 0.

22.9.4 ProxSense Interrupt and Status Register (PXS_ISR)

Address offset: 0x04
Reset value: 0x00

7	6	5	4	3	2	1	0
EOCF	FCCF	NOISEDETF	CIPF	SYNCPF	SYNC_OVRF	Reserved	
rc	rc	rc	r	r	r		

Bit 7 **EOCF**: End of conversion flag

0: No end of conversion has occurred

1: End of conversion has occurred

This status bit is set by hardware when each of the enabled Rx channels has finished converting and the results are ready.

EOCF is cleared by hardware when a conversion is launched (when CIPF becomes ‘1’) or when PXSEN = 0.

If EOCITEN = 1, the ProxSense interrupt occurs when EOCF transitions to 1. EOCF must be cleared by software before exiting the interrupt service routine.

Note: When clearing this flag, it is recommended to write 0x60 to this register to avoid accidentally erasing the other flag bits. The BRES instruction is discouraged since an event may occur between the read and write.

Bit 6 **FCCF**: First conversion completion flag

0: No first conversion completion interrupt has occurred

1: First conversion completion interrupt has occurred

This status bit is set by hardware the first time a channel conversion is completed after a conversion has started. Subsequent conversion completions on other channels do not cause this bit to be set.

FCCF is cleared by hardware when a conversion is launched (when CIPF becomes ‘1’) or when PXSEN = 0.

If FCCITEN = 1, the ProxSense interrupt occurs when FCCF rises. FCCF must be cleared by software before exiting the interrupt service routine.

Note: When clearing this flag, it is recommended to write 0xA0 to this register to avoid accidentally erasing the other flag bits. The BRES instruction is discouraged since an event may occur between the read and write.

Bit 5 **NOISEDETF**: Noise detection flag

0: No noise detected

1: Noise detected

This status bit is set by hardware if RF noise is detected during a conversion. NOISEDETF is cleared by software.

Note: When clearing this flag, it is recommended to write 0xC0 to this register to avoid accidentally erasing the other flag bits. The BRES instruction is discouraged since an event may occur between the read and write.

Bit 4 **CIPF**: Conversion in progress flag

0: No conversion is in progress

1: A conversion is in progress

This status bit is set by hardware when the START bit is set and it is cleared by hardware when the conversion is completed. CIPF is read-only.

Note: Many control registers are locked (read-only) when CIPF = 1.

Note: If PXSEN is set to 0 (by software) during a conversion, the conversion is stopped. In this case, CIPF remains at ‘1’ for as long as 8 µs after which it transitions to ‘0’.

Bit 3 **SYNCPF**: Synchronization (SYNC) pending flag

0: The ProxSense peripheral does not wait for an active edge on the PXS_TRIG input

1: The ProxSense peripheral waits for an active edge on the PXS_TRIG input

This status bit is set by hardware when the START bit is set and when the SYNC feature is enabled (SYNCEN = 1 and PXSEN = 1). It is cleared by hardware when the selected edge (as specified by SYNCEDGE) occurs, at which time a conversion is launched.

Bit 2 **SYNC_OVRF**: Synchronization (SYNC) overflow flag

0: No SYNC overflow occurs

1: A SYNC overflow occurs

There is a “SYNC overflow” if the selected edge occurs on PXS_TRIG pin while SYNCEN = 1, SYNCPF = 0, and PXSEN = 1. Such edges are ignored and the status bit SYNC_OVRF is set. There is not necessarily a problem when SYNC_OVRF = 1 as this just indicates that no conversion was launched on an PXS_TRIG active edge.

Bits 1:0 Reserved

22.9.5 ProxSense Clock Control Register 1 (PXS_CKCR1)

Address offset: 0x06
Reset value: 0x30

7	6	5	4	3	2	1	0
Reserved	PRESC[2:0]			Reserved		ANADEAD	INCPHASE
	rw					rw	rw

Bit 7 Reserved

Bits 6:4 **PRESC[2:0]**: Frequency selection for ProxSense clock

000: 125 kHz
001: 250 kHz
010: 500 kHz
011: 1 MHz (default after reset)
100: 2 MHz
101: 4 MHz
110: 8 MHz
111: 16 MHz

Note: These bits can be modified only when PXSEN = 0.

Bits 3:2 Reserved

Bit 1 **ANADEAD***:* Ensures the deadtime (non-overlap interval) between the UP and PASS phases using a short analog delay rather than a half-cycle deadtime.

0: The deadtime between the UP and PASS phases is one half of a ProxSense clock cycle.

1: The deadtime between the UP and PASS phases is about 10 ns, and the length of the UP phase is increased by one cycle.

Note: This bit can be modified only when CIPF = 0.

Bit 0 **INCPHASE***:* Increases the length of each of the UP and PASS phases by one half of a clock cycle.
0: The charge-transfer phase lengths are as follows:
UP phase: 0.5 + UPLEN + ANADEAD cycles of the ProxSense clock
PASS phase: 0.5 + PASSLEN cycles of the ProxSense clock
1: The charge-transfer phase lengths are one half cycle longer:
UP phase: 1 + UPLEN + ANADEAD cycles of the ProxSense clock
PASS phase: 1 + PASSLEN cycles of the ProxSense clock
Note: This bit can be modified only when CIPF = 0.

22.9.6 ProxSense Clock Control Register 2 (PXS_CKCR2)

Address offset: 0x07
Reset value: 0x11

7	6	5	4	3	2	1	0
Reserved	UPLEN[2:0]			Reserved	PASSLEN[2:0]		
	rw				rw		

Bit 7 Reserved

Bits 6:4 **UPLEN[2:0]**: Length of UP phase(1)
The number of cycles in the UP phase is: 0.5 + UPLEN + (0.5 * INCPHASE) + ANADEAD.
Note: These bits can be modified only when PXSEN = 0.

Bit 3 Reserved

Bits 2:0 **PASSLEN[2:0]:** Length of PASS phase(1)
The number of cycles in the PASS phase is: 0.5 + PASSLEN + (0.5 * INCPHASE).
Note: These bits can be modified only when PXSEN = 0.

1. For UPLEN and PASSLEN, if the value is less than or equal to 4, UP phase = 0.5+UPLEN+(0.5*INCPHASE)+ANADEAD; if the value is greater than 4, UP phase = 0.5+2^(UPLEN-2)+(0.5*INCPHASE)+ANADEAD; So, if UPLEN = 5, UP phase = 0.5+8+(0.5*INCPHASE)+ANADEAD.

22.9.7 Receiver Enable Register for Rx[9:8] (PXS_RXENRH)

Address offset: 0x08
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						RXEN9	RXEN8
						rw	rw

22.9.8 Receiver Enable Register for Rx[7:0] (PXS_RXENRL)

Address offset: 0x09
Reset value: 0x00

7	6	5	4	3	2	1	0
RXEN7	RXEN6	RXEN5	RXEN4	RXEN3	RXEN2	RXEN1	RXEN0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved

Bits 9:0 **RXEN*n***: Enable receiver channel *n*

0: Rx channel *n* is disabled and its inactive state is determined by RXINS*n*.

1: Rx channel *n* is enabled and its functional mode is determined by RXCR1/2/3_*n*.

Note: These bits can be modified only when CIPF = 0.

22.9.9 Receiver Control Register 1 for Rx[9:8] (PXS_RXCR1H)

Address offset: 0x0A
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						RXCR1_9	RXCR1_8
						rw	rw

22.9.10 Receiver Control Register 1 for Rx[7:0] (PXS_RXCR1L)

Address offset: 0x0B
Reset value: 0x00

7	6	5	4	3	2	1	0
RXCR1_7	RXCR1_6	RXCR1_5	RXCR1_4	RXCR1_3	RXCR1_2	RXCR1_1	RXCR1_0
rw	rw	rw	rw	rw	rw	rw	rw

22.9.11 Receiver Control Register 2 for Rx[9:8] (PXS_RXCR2H)

Address offset: 0x0C
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						RXCR2_9	RXCR2_8
						rw	rw

22.9.12 Receiver Control Register 2 for Rx[7:0] (PXS_RXCR2L)

Address offset: 0x0D
Reset value: 0x00

7	6	5	4	3	2	1	0
RXCR2_7	RXCR2_6	RXCR2_5	RXCR2_4	RXCR2_3	RXCR2_2	RXCR2_1	RXCR2_0
rw	rw	rw	rw	rw	rw	rw	rw

22.9.13 Receiver Control Register 3 for Rx[9:8] (PXS_RXCR3H)

Address offset: 0x0E
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						RXCR3_9	RXCR3_8
						rw	rw

22.9.14 Receiver Control Register 3 for Rx[7:0] (PXS_RXCR3L)

Address offset: 0x0F
Reset value: 0x00

7	6	5	4	3	2	1	0
RXCR3_7	RXCR3_6	RXCR3_5	RXCR3_4	RXCR3_3	RXCR3_2	RXCR3_1	RXCR3_0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved

Bits 9:0 **RXCR1/2/3_*n***: Receiver channel *n* Conversion mode
The following modes take effect when RXEN*n* = 1 and CIPF = 1.
011: Receiver channel *n* is in Projected transmitting mode (Tx)
111: Receiver channel *n* is in Projected receiving mode (Rx)
These bits can be modified only when CIPF = 0.

22.9.15 Receiver Inactive State Register for Rx[9:8] (PXS_RXINSRH)

Address offset: 0x12
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						RXINS9	RXINS8
						rw	rw

22.9.16 Receiver Inactive State Register for Rx[7:0] (PXS_RXINSRL)

Address offset: 0x13
Reset value: 0x00

7	6	5	4	3	2	1	0
RXINS7	RXINS6	RXINS5	RXINS4	RXINS3	RXINS2	RXINS1	RXINS0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved

Bits 9:0 **RXINS*n***: Receiver channel *n* inactive state selection

This bit is valid when the corresponding receiver channel is disabled (RXEN*n* = 0). It also applies to a channel when the group (A or B) to which it belongs is not selected (for example, it applies to all Rx ‘B’ channels when RXGROUP = 0).

0: Receiver channel *n* is driven to VSS when disabled.

1: Receiver channel *n* is high impedance when disabled.

22.9.17 Transmit Enable Register for Tx[15:8] (PXS_TXENRH)

Address offset: 0x16
Reset value: 0x00

7	6	5	4	3	2	1	0
TXEN15	TXEN14	TXEN13	TXEN12	TXEN11	TXEN10	TXEN9	TXEN8
rw	rw	rw	rw	rw	rw	rw	rw

22.9.18 Transmit Enable Register for Tx[7:0] (PXS_TXENRL)

Address offset: 0x17
Reset value: 0x00

7	6	5	4	3	2	1	0
TXEN7	TXEN6	TXEN5	TXEN4	TXEN3	TXEN2	TXEN1	TXEN0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **TXEN*m***: Transmit output *m* function enable

0: The alternate function for Transmit output *m* is disabled and the corresponding pin becomes a general-purpose IO (GPIO) or can be used by other alternate functions.

1: The alternate function for Transmit output *m* is enabled

Note: When PXSEN = 0, all transmit alternate functions are disabled.

Note: These bits can be modified only when CIPF = 0.

Note: All transmitters can be individually enabled to operate simultaneously.

22.9.19 Maximum Counter Value Register for Rx[9:8] (PXS_MAXRH)

Address offset: 0x1A
Reset value: 0xFF

7	6	5	4	3	2	1	0
MAX[15:8]							
rw							

22.9.20 Maximum Counter Value Register for Rx[7:0] (PXS_MAXRL)

Address offset: 0x1B
Reset value: 0xFF

7	6	5	4	3	2	1	0
MAX[7:0]							
rw							

Bits 15:10 Reserved

Bits 7:0 **MAX[15:0]**: 16-bit value indicating the maximum allowed value for the conversion data. When the conversion data of a receiver channel reaches this value, the conversion stops. The conversion data (PXS_RX*n*CNTR) is 0x0000 for those channels which did not reach the threshold voltage (V_{THR}) before reaching the maximum count and their corresponding data-valid bit (VALID*n*) is ‘0’ in PXS_RXSR.

Note: These registers can be modified only when CIPF = 0.

22.9.21 Maximum Counter Enable Register for Rx[9:8] (PXS_MAXENRH)

Address offset: 0x1C
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						MAXEN9	MAXEN8
						rw	rw

22.9.22 Maximum Counter Enable Register for Rx[7:0] (PXS_MAXENRL)

Address offset: 0x1D
Reset value: 0x00

7	6	5	4	3	2	1	0
MAXEN7	MAXEN6	MAXEN5	MAXEN4	MAXEN3	MAXEN2	MAXEN1	MAXEN0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved

Bits 9:0 **MAXEN*n***: Enabled maximum counter limit for Receive channel *n*

0: RX*n*CNTR is not limited: it is set to the counter value when it reaches the threshold voltage (V_{THR}), or to '0' if V_{THR} is never attained.

1: RX*n*CNTR is limited to MAX[15:0]. If Receive channel *n* does not reach the threshold voltage (V_{THR}) by the time its counter reaches MAX, the conversion is halted and RX*n*CNTR is read as 0x0000.

Note: These bits can be modified only when CIPF = 0.

22.9.23 Receiver Status Register for Rx[9:8] (PXS_RXSRH)

Address offset: 0x1E
Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						VALID9	VALID8
						r	r

22.9.24 Receiver Status Register for Rx[7:0] (PXS_RXSRL)

Address offset: 0x1F
Reset value: 0x00

7	6	5	4	3	2	1	0
VALID7	VALID6	VALID5	VALID4	VALID3	VALID2	VALID1	VALID0
r	r	r	r	r	r	r	r

Bits 15:10 Reserved

Bits 9:0 **VALID*n***: Valid bit for conversion data for receiver channel *n* is valid

0: RX*n*CNTR is ‘0’ and not valid

1: RX*n*CNTR is valid

As soon as the voltage on a Cs capacitor has reached the threshold voltage (V_{THR}), its data register is updated and the corresponding VALID*n* bit is set.

PXS_RXSR is cleared at the start of each conversion (when START transitions from ‘0’ to ‘1’). PXS_RXSR is also cleared when PXSEN = 0.

VALID*n* remains cleared when the channel is disabled (RXEN*n* = 0), or when it is in Projected transmitting mode (RXCR1/2/3_*n* is ‘011’). It also remains cleared if the corresponding channel’s data reaches 65535 (0xFFFF) if MAXEN*n* is cleared or exceeds MAX[15:0] and MAXEN*n* is ‘1’.

22.9.25 Counter Register for bits [15:8] of Receiver Channel n (PXS_RX*n*CNTRH)

Address offset: 0x20 - 0x32 (even addresses)
Reset values: 0x00

7	6	5	4	3	2	1	0
RX*n*CNT[15:8]							
r							

22.9.26 Counter Register for bits [7:0] of Receiver Channel n (PXS_RX*n*CNTRL)

Address offset: 0x21 - 0x33 (odd addresses)
Reset values: 0x00

7	6	5	4	3	2	1	0
RX*n*CNT[7:0]							
r							

Bits 15:0 **RX*n*CNT[15:0]**: 16-bit counter value resulting from a measurement on receiver channel *n*.

Note: This value is valid when VALID = 1 in PXS_RXSR, otherwise it is read at 0x0000.

Note: This value is cleared when PXSEN = 0.

22.9.27 Receiver 0 to 9 Sampling Capacitor (C_S) Size Selection Registers 0-9 (PXS_RX*n*CSSELR)

Address offsets: 0x40 - 0x49
Reset values: 0x00

7	6	5	4	3	2	1	0
Reserved			RX*n*CSSEL[4:0]				
			rw				

Bits 7:5 Reserved

Bits 4:0 **RX*n*CSSEL[4:0]**: Size selection for the internal sampling capacitor (Cs) of Receiver Channel *n*.
0x00 - 0x1F: TBD
This register can be modified only when CIPF = 0.

22.9.28 Receiver 0- to 9 Electrode Parasitic Compensation Capacitor (EPCC) Size Selection Registers 0-9 (PXS_RX*n*EPCCSELR)

Address offsets: 0x50 - 0x59
Reset values: 0x00

7	6	5	4	3	2	1	0
RX*n*EPCC[7:0]							
rw							

Bits 7:0 **RX*n*EPCC[7:0]**: Size selection for the internal electrode parasitic compensation capacitor (EPCC) of Receiver Channel *n*.
0x00: EPCC disabled
0x01 - 0xFF: EPCC capacitor size for receiver channel *n* is:
RX*n*EPCC * 0.0625 pF

Note: This register can be modified only when CIPF = 0.

22.9.29 ProxSense register map and reset values

Table 56. ProxSense register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	PXS_CR1 Reset value	PXSEN 0	START 0	LOW_ POWER 0	- 00000				
0x01	PXS_CR2 Reset value	EOCITEN 0	FCCITEN 0	NOISE-DETEN 0	- 0	RXGROUP 0	RXCOUPLING 0	SYNCEN 0	SYNCEDGE 0
0x02	PXS_CR3 Reset value	STAB[1:0] 00		BIAS[1:0] 00		VTHR[3:0] 0100			
0x04	PXS_ISR Reset value	EOCF 0	FCCF 0	NOISE-DETF 0	CIPF 0	SYNCPF 0	SYNC_ OVRF 0	- 00	
0x06	PXS_CKCR1 Reset value	- 0	PRESC[2:0] 011			- 00		ANADEAD 0	INCPHASE 0
0x07	PXS_CKCR2 Reset value	- 0	UPLEN[2:0] 001			- 0	PASSLEN[2:0] 001		
0x08	PXS_RXENRH Reset value	- 000000						RXEN9 0	RXEN8 0
0x09	PXS_RXENRL Reset value	RXEN7 0	RXEN6 0	RXEN5 0	RXEN4 0	RXEN3 0	RXEN2 0	RXEN1 0	RXEN0 0
0x0A	PXS_RXCR1H Reset value	- 000000						RXCR1_9 0	RXCR1_8 0
0x0B	PXS_RXCR1L Reset value	RXCR1_7 0	RXCR1_6 0	RXCR1_5 0	RXCR1_4 0	RXCR1_3 0	RXCR1_2 0	RXCR1_1 0	RXCR1_0 0
0x0C	PXS_RXCR2H Reset value	- 000000						RXCR2_9 0	RXCR2_8 0
0x0D	PXS_RXCR2L Reset value	RXCR2_7 0	RXCR2_6 0	RXCR2_5 0	RXCR2_4 0	RXCR2_3 0	RXCR2_2 0	RXCR2_1 0	RXCR2_0 0
0x0E	PXS_RXCR3H Reset value	- 000000						RXCR3_9 0	RXCR3_8 0
0x0F	PXS_RXCR3L Reset value	RXCR3_7 0	RXCR3_6 0	RXCR3_5 0	RXCR3_4 0	RXCR3_3 0	RXCR3_2 0	RXCR3_1 0	RXCR3_0 0
0x12	PXS_RXINSRH Reset value	- 000000						RXINS9 0	RXINS8 0
0x13	PXS_RXINSRL Reset value	RXINS7 0	RXINS6 0	RXINS5 0	RXINS4 0	RXINS3 0	RXINS2 0	RXINS1 0	RXINS0 0
0x16	PXS_TXENRH Reset value	TXEN15 0	TXEN14 0	TXEN13 0	TXEN12 0	TXEN11 0	TXEN10 0	TXEN9 0	TXEN8 0
0x17	PXS_TXENRL Reset value	TXEN7 0	TXEN6 0	TXEN5 0	TXEN4 0	TXEN3 0	TXEN2 0	TXEN1 0	TXEN0 0
0x1A	PXS_MAXRH Reset value	MAX[15:8] 11111111							
0x1B	PXS_MAXRL Reset value	MAX[7:0] 11111111							
0x1C	PXS_MAXENRH Reset value	- 000000						MAXEN9 0	MAXEN8 0
0x1D	PXS_MAXENRL Reset value	MAXEN7 0	MAXEN6 0	MAXEN5 0	MAXEN4 0	MAXEN3 0	MAXEN2 0	MAXEN1 0	MAXEN0 0

Table 56. ProxSense register map (continued)

Address offset	Register name	7	6	5	4	3	2	1	0
0x1E	PXS_RXSRH Reset Value	- 000000						VALID9 0	VALID8 0
0x1F	PXS_RXSRL Reset Value	VALID7 0	VALID6 0	VALID5 0	VALID4 0	VALID3 0	VALID2 0	VALID1 0	VALID0 0
0x20	PXS_RX0CNTRH Reset value	RX0CNT[15:8] 00000000							
0x21	PXS_RX0CNTRL Reset value	RX0CNT[7:0] 00000000							
0x22	PXS_RX*n*CNTRH Reset value	RX*n*CNT[15:8] 00000000							
0x23	PXS_RX*n*CNTRL Reset value	RX*n*CNT[7:0] 00000000							
...	...	...							
0x32	PXS_RX9CNTRH Reset value	RX9CNT[15:8] 00000000							
0x33	PXS_RX9CNTRL Reset value	RX9CNT[7:0] 00000000							
0x40	PXS_RX0CSSELR Reset value	- 000			RX0CSSEL[4:0] 00000				
0x41	PXS_RX*n*CSSELR Reset value	- 000			RX*n*CSSEL[4:0] 00000				
...	...	...							
0x49	PXS_RX9CSSELR Reset value	- 000			RX9CSSEL[4:0] 00000				
0x50	PXS_RX0EPCCSELR Reset value	RX0EPCC[7:0] 00000000							
0x51	PXS_RX*n*EPCCSELR Reset value	RX*n*EPCC[7:0] 00000000							
...	...	...							
0x59	PXS_RX9EPCCSELR Reset value	RX9EPCC[7:0] 00000000							

23 Revision history

Table 57. Document revision history

Date	Revision	Changes
28-Oct-2011	1	Initial release
20-Mar-2012	2	Changed title to “STM8TL5xxx microcontroller family” to cover STM8T52x and STM8TL53x devices Added PCODE description to *Section 4.4: Memory organization on page 26*.
08-Oct-2013	3	Replaced CLK_CCOR reset value to 0x10. Updated CCOSLP0 bit reset value to 1 in *Table 16: CLK register map and reset values*. Added *Chapter 11: System configuration controller (SYSCFG)*. Updated bit3 of SPI_CR1 register in *Section 20.5: SPI register map and reset values*. Updated Internal signal name related to the timer input filtered edge detector and the description of TRGO in *Table 33: Glossary of internal timer signals*. Updated the *Figure 19: TIMx general block diagram* and *Figure 30: Clock/trigger controller block diagram*. Added sub section *SCL master clock generation* in section *Section 19.4.2: I^2C master mode*. Updated FREQ bit description in *Section 19.7.3: Frequency register (I2C_FREQR)*. Updated TRISE bit description in *Section 19.7.14: TRISE register (I2C_TRISER)*. Updated the notes of CCR bits in both *Section 19.7.12: Clock control register low (I2C_CCRL)* and *Section 19.7.13: Clock control register high (I2C_CCRH)*. Updated range of FREQ bits in *Section 19.7.3: Frequency register (I2C_FREQR)*. Updated the disclaimer.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com