

## STM8S Series and STM8AF Series 8-bit microcontrollers

## Introduction

This reference manual provides complete information for application developers on how to use STM8S Series and STM8AF Series microcontroller memory and peripherals.

The STM8AF Series of microcontrollers is designed for automotive applications, with different memory densities, packages and peripherals:

- The low-density STM8AF devices are the STM8AF6223/26 with 8 Kbytes of Flash memory.
- The medium-density STM8AF devices are the STM8AF624x and STM8AF6266/68 microcontrollers with 16 to 32 Kbytes of Flash memory.
- The high-density STM8AF devices are the STM8AF52xx and STM8AF6269/8x/Ax microcontrollers with 32 to 128 Kbytes of Flash memory.

The STM8S Series of microcontrollers is designed for general purpose applications, with different memory densities, packages and peripherals.

- The value-line low-density STM8S devices are the STM8S001xx/STM8S003xx microcontrollers with 8 Kbytes of Flash memory.
- The value-line medium-density STM8S devices are the STM8S005xx microcontrollers with 32 Kbytes of Flash memory.
- The value-line high-density STM8S devices are the STM8S007xx microcontrollers with 64 Kbytes of Flash memory.
- The access-line low-density STM8S devices are the STM8S103xx and STM8S903xx microcontrollers with 8 Kbytes of Flash memory.
- The access-line medium-density STM8S devices are the STM8S105xx microcontrollers with 16 to 32 Kbytes of Flash memory.
- The performance-line high-density STM8S devices are the STM8S207xx and STM8S208xx microcontrollers with 32 to 128 Kbytes of Flash memory.

Refer to the product datasheet for ordering information, pin description, mechanical and electrical device characteristics, and for the complete list of available peripherals.

## Reference documents

- For information on programming, erasing and protection of the internal Flash memory please refer to the STM8S Series and STM8AF Series Flash programming manual (PM0051), and to the STM8 SWIM communication protocol and debug module user manual (UM0470).
- For information on the STM8 core, refer to STM8 CPU programming manual (PM0044).
- The bootloader user manual (UM0560) describes the usage of the integrated ROM bootloader.

# Contents

<b>1</b>	<b>Central processing unit (CPU) . . . . .</b>	<b>23</b>
1.1	CPU introduction . . . . .	23
1.2	CPU registers . . . . .	23
1.2.1	Description of CPU registers . . . . .	23
1.2.2	STM8 CPU register map . . . . .	27
1.3	Global configuration register (CFG_GCR) . . . . .	27
1.3.1	Activation level . . . . .	27
1.3.2	SWIM disable . . . . .	27
1.3.3	Description of global configuration register (CFG_GCR) . . . . .	28
1.3.4	Global configuration register map and reset values . . . . .	28
<b>2</b>	<b>Boot ROM . . . . .</b>	<b>29</b>
<b>3</b>	<b>Memory and register map . . . . .</b>	<b>30</b>
3.1	Memory layout . . . . .	30
3.1.1	Memory map . . . . .	30
3.1.2	Stack handling . . . . .	31
3.2	Register description abbreviations . . . . .	33
<b>4</b>	<b>Flash program memory and data EEPROM . . . . .</b>	<b>34</b>
4.1	Flash and EEPROM introduction . . . . .	34
4.2	Flash and EEPROM glossary . . . . .	34
4.3	Main Flash memory features . . . . .	35
4.4	Memory organization . . . . .	36
4.4.1	STM8S and STM8AF memory organization . . . . .	36
4.4.2	Memory access/ wait state configuration . . . . .	40
4.4.3	User boot area (UBC) . . . . .	40
4.4.4	Data EEPROM (DATA) . . . . .	43
4.4.5	Main program area . . . . .	43
4.4.6	Option bytes . . . . .	43
4.5	Memory protection . . . . .	44
4.5.1	Readout protection . . . . .	44
4.5.2	Memory access security system (MASS) . . . . .	44

4.5.3	Enabling write access to option bytes . . . . .	45
4.6	Memory programming . . . . .	46
4.6.1	Read-while-write (RWW) . . . . .	46
4.6.2	Byte programming . . . . .	46
4.6.3	Word programming . . . . .	47
4.6.4	Block programming . . . . .	47
4.6.5	Option byte programming . . . . .	49
4.7	ICP (in-circuit programming) and IAP (in-application programming) . . . . .	49
4.8	Flash registers . . . . .	51
4.8.1	Flash control register 1 (FLASH_CR1) . . . . .	51
4.8.2	Flash control register 2 (FLASH_CR2) . . . . .	52
4.8.3	Flash complementary control register 2 (FLASH_NCR2) . . . . .	53
4.8.4	Flash protection register (FLASH_FPR) . . . . .	54
4.8.5	Flash protection register (FLASH_NFPR) . . . . .	54
4.8.6	Flash program memory unprotecting key register (FLASH_PUKR) . . . . .	54
4.8.7	Data EEPROM unprotection key register (FLASH_DUKR) . . . . .	55
4.8.8	Flash status register (FLASH_IAPSR) . . . . .	55
4.8.9	Flash register map and reset values . . . . .	56
<b>5</b>	<b>Single wire interface module (SWIM) and debug module (DM) . . . . .</b>	<b>57</b>
5.1	SWIM and DM introduction . . . . .	57
5.2	SWIM main features . . . . .	57
5.3	SWIM modes . . . . .	57
<b>6</b>	<b>Interrupt controller (ITC) . . . . .</b>	<b>59</b>
6.1	ITC introduction . . . . .	59
6.2	Interrupt masking and processing flow . . . . .	59
6.2.1	Servicing pending interrupts . . . . .	60
6.2.2	Interrupt sources . . . . .	61
6.3	Interrupts and low power modes . . . . .	63
6.4	Activation level/low power mode control . . . . .	63
6.5	Concurrent and nested interrupt management . . . . .	64
6.5.1	Concurrent interrupt management mode . . . . .	64
6.5.2	Nested interrupt management mode . . . . .	65
6.6	External interrupts . . . . .	66
6.7	Interrupt instructions . . . . .	66

---

6.8	Interrupt mapping . . . . .	67
6.9	ITC and EXTI registers . . . . .	68
6.9.1	CPU condition code register interrupt bits (CCR) . . . . .	68
6.9.2	Software priority register x (ITC_SPRx) . . . . .	69
6.9.3	External interrupt control register 1 (EXTI_CR1) . . . . .	70
6.9.4	External interrupt control register 1 (EXTI_CR2) . . . . .	71
6.9.5	ITC and EXTI register map and reset values . . . . .	72
7	<b>Power supply . . . . .</b>	<b>73</b>
8	<b>Reset (RST) . . . . .</b>	<b>74</b>
8.1	“Reset state” and “under reset” definitions . . . . .	74
8.2	Reset circuit description . . . . .	74
8.3	Internal reset sources . . . . .	75
8.3.1	Power-on reset (POR) and brown-out reset (BOR) . . . . .	75
8.3.2	Watchdog reset . . . . .	75
8.3.3	Software reset . . . . .	76
8.3.4	SWIM reset . . . . .	76
8.3.5	Illegal opcode reset . . . . .	76
8.3.6	EMC reset . . . . .	76
8.4	RST register description . . . . .	77
8.4.1	Reset status register (RST_SR) . . . . .	77
8.5	RST register map . . . . .	77
9	<b>Clock control (CLK) . . . . .</b>	<b>78</b>
9.1	Master clock sources . . . . .	80
9.1.1	HSE (high-speed external) clock signal . . . . .	80
9.1.2	HSI (high-speed internal) clock signal . . . . .	81
9.1.3	LSI . . . . .	82
9.2	Master clock switching . . . . .	83
9.2.1	System startup . . . . .	83
9.2.2	Master clock switching procedures . . . . .	83
9.3	Low-speed clock selection . . . . .	86
9.4	CPU clock-divider . . . . .	86
9.5	Peripheral clock-gating (PCG) . . . . .	87
9.6	Clock security system (CSS) . . . . .	88

9.7	Clock-out capability (CCO) . . . . .	89
9.8	CLK interrupts . . . . .	89
9.9	CLK register description . . . . .	90
9.9.1	Internal clock register (CLK_ICKR) . . . . .	90
9.9.2	External clock register (CLK_ECKR) . . . . .	91
9.9.3	Clock master status register (CLK_CMSR) . . . . .	92
9.9.4	Clock master switch register (CLK_SWR) . . . . .	92
9.9.5	Switch control register (CLK_SWCR) . . . . .	93
9.9.6	Clock divider register (CLK_CKDIVR) . . . . .	94
9.9.7	Peripheral clock gating register 1 (CLK_PCKENR1) . . . . .	95
9.9.8	Peripheral clock gating register 2 (CLK_PCKENR2) . . . . .	96
9.9.9	Clock security system register (CLK_CSSR) . . . . .	97
9.9.10	Configurable clock output register (CLK_CCOR) . . . . .	98
9.9.11	HSI clock calibration trimming register (CLK_HSITRIMR) . . . . .	99
9.9.12	SWIM clock control register (CLK_SWIMCCR) . . . . .	99
9.10	CLK register map and reset values . . . . .	100
<b>10</b>	<b>Power management . . . . .</b>	<b>101</b>
10.1	General considerations . . . . .	101
10.1.1	Clock management for low consumption . . . . .	102
10.2	Low power modes . . . . .	102
10.2.1	Wait mode . . . . .	103
10.2.2	Halt mode . . . . .	103
10.2.3	Active-halt modes . . . . .	103
10.3	Additional analog power controls . . . . .	104
10.3.1	Fast Flash wakeup from Halt mode . . . . .	104
10.3.2	Very low Flash consumption in Active-halt mode . . . . .	104
<b>11</b>	<b>General purpose I/O ports (GPIO) . . . . .</b>	<b>105</b>
11.1	Introduction . . . . .	105
11.2	GPIO main features . . . . .	105
11.3	Port configuration and usage . . . . .	106
11.3.1	Input modes . . . . .	107
11.3.2	Output modes . . . . .	108
11.4	Reset configuration . . . . .	108
11.5	Unused I/O pins . . . . .	108

---

11.6	Low power modes .....	108
11.7	Input mode details .....	109
11.7.1	Alternate function input .....	109
11.7.2	Interrupt capability .....	109
11.7.3	Analog channels .....	109
11.7.4	Schmitt trigger .....	110
11.7.5	Analog function .....	110
11.8	Output mode details .....	110
11.8.1	Alternate function output .....	110
11.8.2	Slope control .....	110
11.9	GPIO registers .....	111
11.9.1	Port x output data register (Px_ODR) .....	111
11.9.2	Port x pin input register (Px_IDR) .....	111
11.9.3	Port x data direction register (Px_DDR) .....	112
11.9.4	Port x control register 1 (Px_CR1) .....	112
11.9.5	Port x control register 2 (Px_CR2) .....	113
11.9.6	GPIO register map and reset values .....	113
<b>12</b>	<b>Auto-wakeup (AWU) .....</b>	<b>114</b>
12.1	AWU introduction .....	114
12.2	LSI clock measurement .....	114
12.3	AWU functional description .....	115
12.3.1	AWU operation .....	115
12.3.2	Time base selection .....	116
12.3.3	LSI clock frequency measurement .....	117
12.4	AWU registers .....	118
12.4.1	Control/status register (AWU_CSR) .....	118
12.4.2	Asynchronous prescaler register (AWU(APR)) .....	119
12.4.3	Timebase selection register (AWU_TBR) .....	119
12.4.4	AWU register map and reset values .....	120
<b>13</b>	<b>Beeper (BEEP) .....</b>	<b>121</b>
13.1	Beeper introduction .....	121
13.2	Beeper functional description .....	121
13.2.1	Beeper operation .....	121
13.2.2	Beeper calibration .....	122

13.3	Beeper registers . . . . .	122
13.3.1	Beeper control/status register (BEEP_CSR) . . . . .	122
13.3.2	Beeper register map and reset values . . . . .	123
<b>14</b>	<b>Independent watchdog (IWDG) . . . . .</b>	<b>124</b>
14.1	IWDG introduction . . . . .	124
14.2	IWDG functional description . . . . .	124
14.3	IWDG registers . . . . .	126
14.3.1	Key register (IWDG_KR) . . . . .	126
14.3.2	Prescaler register (IWDG_PR) . . . . .	126
14.3.3	Reload register (IWDG_RLR) . . . . .	127
14.3.4	IWDG register map and reset values . . . . .	127
<b>15</b>	<b>Window watchdog (WWDG) . . . . .</b>	<b>128</b>
15.1	WWDG introduction . . . . .	128
15.2	WWDG main features . . . . .	128
15.3	WWDG functional description . . . . .	128
15.4	How to program the watchdog timeout . . . . .	130
15.5	WWDG low power modes . . . . .	131
15.6	Hardware watchdog option . . . . .	132
15.7	Using Halt mode with the WWDG (WWDGHALT option) . . . . .	132
15.8	WWDG interrupts . . . . .	132
15.9	WWDG registers . . . . .	132
15.9.1	Control register (WWDG_CR) . . . . .	132
15.9.2	Window register (WWDG_WR) . . . . .	133
15.10	Window watchdog register map and reset values . . . . .	133
<b>16</b>	<b>Timer overview . . . . .</b>	<b>134</b>
16.1	Timer feature comparison . . . . .	135
16.2	Glossary of timer signal names . . . . .	136
<b>17</b>	<b>16-bit advanced control timer (TIM1) . . . . .</b>	<b>138</b>
17.1	TIM1 introduction . . . . .	138
17.2	TIM1 main features . . . . .	139
17.3	TIM1 time base unit . . . . .	141

---

17.3.1	Reading and writing to the 16-bit counter . . . . .	142
17.3.2	Write sequence for 16-bit TIM1_ARR register . . . . .	142
17.3.3	Prescaler . . . . .	142
17.3.4	Up-counting mode . . . . .	143
17.3.5	Down-counting mode . . . . .	145
17.3.6	Center-aligned mode (up/down counting) . . . . .	147
17.3.7	Repetition down-counter . . . . .	149
17.4	TIM1 clock/trigger controller . . . . .	151
17.4.1	Prescaler clock (CK_PSC) . . . . .	151
17.4.2	Internal clock source (fMASTER) . . . . .	152
17.4.3	External clock source mode 1 . . . . .	152
17.4.4	External clock source mode 2 . . . . .	154
17.4.5	Trigger synchronization . . . . .	155
17.4.6	Synchronization between TIM1, TIM5 and TIM6 timers . . . . .	159
17.5	TIM1 capture/compare channels . . . . .	166
17.5.1	Write sequence for 16-bit TIM1_CCRI registers . . . . .	167
17.5.2	Input stage . . . . .	168
17.5.3	Input capture mode . . . . .	169
17.5.4	Output stage . . . . .	171
17.5.5	Forced output mode . . . . .	172
17.5.6	Output compare mode . . . . .	172
17.5.7	PWM mode . . . . .	174
17.5.8	Using the break function . . . . .	181
17.5.9	Clearing the OCiREF signal on an external event . . . . .	184
17.5.10	Encoder interface mode . . . . .	185
17.6	TIM1 interrupts . . . . .	187
17.7	TIM1 registers . . . . .	188
17.7.1	Control register 1 (TIM1_CR1) . . . . .	188
17.7.2	Control register 2 (TIM1_CR2) . . . . .	190
17.7.3	Slave mode control register (TIM1_SMCR) . . . . .	191
17.7.4	External trigger register (TIM1_ETR) . . . . .	192
17.7.5	Interrupt enable register (TIM1_IER) . . . . .	194
17.7.6	Status register 1 (TIM1_SR1) . . . . .	195
17.7.7	Status register 2 (TIM1_SR2) . . . . .	196
17.7.8	Event generation register (TIM1_EGR) . . . . .	197
17.7.9	Capture/compare mode register 1 (TIM1_CCMR1) . . . . .	198
17.7.10	Capture/compare mode register 2 (TIM1_CCMR2) . . . . .	202

17.7.11	Capture/compare mode register 3 (TIM1_CCMR3) . . . . .	203
17.7.12	Capture/compare mode register 4 (TIM1_CCMR4) . . . . .	204
17.7.13	Capture/compare enable register 1 (TIM1_CCER1) . . . . .	205
17.7.14	Capture/compare enable register 2 (TIM1_CCER2) . . . . .	208
17.7.15	Counter high (TIM1_CNTRH) . . . . .	208
17.7.16	Counter low (TIM1_CNTRL) . . . . .	209
17.7.17	Prescaler high (TIM1_PSCRH) . . . . .	209
17.7.18	Prescaler low (TIM1_PSCRL) . . . . .	209
17.7.19	Auto-reload register high (TIM1_ARRH) . . . . .	210
17.7.20	Auto-reload register low (TIM1_ARRL) . . . . .	210
17.7.21	Repetition counter register (TIM1_RCR) . . . . .	210
17.7.22	Capture/compare register 1 high (TIM1_CCR1H) . . . . .	211
17.7.23	Capture/compare register 1 low (TIM1_CCR1L) . . . . .	211
17.7.24	Capture/compare register 2 high (TIM1_CCR2H) . . . . .	212
17.7.25	Capture/compare register 2 low (TIM1_CCR2L) . . . . .	212
17.7.26	Capture/compare register 3 high (TIM1_CCR3H) . . . . .	213
17.7.27	Capture/compare register 3 low (TIM1_CCR3L) . . . . .	213
17.7.28	Capture/compare register 4 high (TIM1_CCR4H) . . . . .	214
17.7.29	Capture/compare register 4 low (TIM1_CCR4L) . . . . .	214
17.7.30	Break register (TIM1_BKR) . . . . .	215
17.7.31	Deadtime register (TIM1_DTR) . . . . .	217
17.7.32	Output idle state register (TIM1_OISR) . . . . .	218
17.7.33	TIM1 register map and reset values . . . . .	219
<b>18</b>	<b>16-bit general purpose timers (TIM2, TIM3, TIM5) . . . . .</b>	<b>222</b>
18.1	TIM2, TIM3 and TIM5 introduction . . . . .	222
18.2	TIM2/TIM3 main features . . . . .	222
18.3	TIM5 main features . . . . .	223
18.4	TIM2/TIM3/TIM5 functional description . . . . .	223
18.4.1	Time base unit . . . . .	224
18.4.2	Clock/trigger controller . . . . .	225
18.4.3	Capture/compare channels . . . . .	226
18.5	TIM2/TIM3/TIM5 interrupts . . . . .	228
18.6	TIM2/TIM3/TIM5 registers . . . . .	229
18.6.1	Control register 1 (TIMx_CR1) . . . . .	229
18.6.2	Control register 2 (TIM5_CR2) . . . . .	230

---

18.6.3	Slave mode control register (TIM5_SMCR) . . . . .	231
18.6.4	Interrupt enable register (TIMx_IER) . . . . .	232
18.6.5	Status register 1 (TIMx_SR1) . . . . .	233
18.6.6	Status register 2 (TIMx_SR2) . . . . .	234
18.6.7	Event generation register (TIMx_EGR) . . . . .	235
18.6.8	Capture/compare mode register 1 (TIMx_CCMR1) . . . . .	236
18.6.9	Capture/compare mode register 2 (TIMx_CCMR2) . . . . .	239
18.6.10	Capture/compare mode register 3 (TIMx_CCMR3) . . . . .	240
18.6.11	Capture/compare enable register 1 (TIMx_CCER1) . . . . .	241
18.6.12	Capture/compare enable register 2 (TIMx_CCER2) . . . . .	242
18.6.13	Counter high (TIMx_CNTRH) . . . . .	242
18.6.14	Counter low (TIMx_CNTRL) . . . . .	243
18.6.15	Prescaler register (TIMx_PSCR) . . . . .	244
18.6.16	Auto-reload register high (TIMx_ARRH) . . . . .	244
18.6.17	Auto-reload register low (TIMx_ARRL) . . . . .	245
18.6.18	Capture/compare register 1 high (TIMx_CCR1H) . . . . .	245
18.6.19	Capture/compare register 1 low (TIMx_CCR1L) . . . . .	246
18.6.20	Capture/compare register 2 high (TIMx_CCR2H) . . . . .	246
18.6.21	Capture/compare register 2 low (TIMx_CCR2L) . . . . .	246
18.6.22	Capture/compare register 3 high (TIMx_CCR3H) . . . . .	247
18.6.23	Capture/compare register 3 low (TIMx_CCR3L) . . . . .	247
<b>19</b>	<b>8-bit basic timer (TIM4, TIM6) . . . . .</b>	<b>253</b>
19.1	TIM4, TIM6 introduction . . . . .	253
19.2	TIM4 main features . . . . .	254
19.3	TIM6 main features . . . . .	254
19.4	TIM4/TIM6 interrupts . . . . .	254
19.5	TIM4/TIM6 clock selection . . . . .	254
19.6	TIM4/TIM6 registers . . . . .	255
19.6.1	Control register 1 (TIMx_CR1) . . . . .	255
19.6.2	Control register 2 (TIM6_CR2) . . . . .	256
19.6.3	Slave mode control register (TIM6_SMCR) . . . . .	256
19.6.4	Interrupt enable register (TIMx_IER) . . . . .	258
19.6.5	Status register 1 (TIMx_SR) . . . . .	258
19.6.6	Event generation register (TIMx_EGR) . . . . .	259
19.6.7	Counter (TIMx_CNTR) . . . . .	259

19.6.8	Prescaler register (TIMx_PSCR) . . . . .	259
19.6.9	Auto-reload register (TIMx_ARR) . . . . .	260
19.6.10	TIM4/TIM6 register map and reset values . . . . .	260
<b>20</b>	<b>Serial peripheral interface (SPI) . . . . .</b>	<b>262</b>
20.1	SPI introduction . . . . .	262
20.2	SPI main features . . . . .	262
20.3	SPI functional description . . . . .	263
20.3.1	SPI general description . . . . .	263
20.3.2	Configuring the SPI in slave mode . . . . .	267
20.3.3	Configuring the SPI master mode . . . . .	267
20.3.4	Configuring the SPI for simplex communications . . . . .	268
20.3.5	Data transmission and reception procedures . . . . .	268
20.3.6	CRC calculation . . . . .	275
20.3.7	Status flags . . . . .	277
20.3.8	Disabling the SPI . . . . .	277
20.3.9	Error flags . . . . .	279
20.3.10	SPI low power modes . . . . .	280
20.3.11	SPI interrupts . . . . .	280
20.4	SPI registers . . . . .	282
20.4.1	SPI control register 1 (SPI_CR1) . . . . .	282
20.4.2	SPI control register 2 (SPI_CR2) . . . . .	283
20.4.3	SPI interrupt control register (SPI_ICR) . . . . .	284
20.4.4	SPI status register (SPI_SR) . . . . .	285
20.4.5	SPI data register (SPI_DR) . . . . .	286
20.4.6	SPI CRC polynomial register (SPI_CRCPR) . . . . .	286
20.4.7	SPI Rx CRC register (SPI_RXCRCR) . . . . .	286
20.4.8	SPI Tx CRC register (SPI_TXCRCR) . . . . .	287
20.5	SPI register map and reset values . . . . .	287
<b>21</b>	<b>Inter-integrated circuit (I2C) interface . . . . .</b>	<b>288</b>
21.1	I2C introduction . . . . .	288
21.2	I2C main features . . . . .	288
21.3	I2C general description . . . . .	289
21.4	I2C functional description . . . . .	291
21.4.1	I2C slave mode . . . . .	291

---

21.4.2	I2C master mode . . . . .	293
21.4.3	Error conditions . . . . .	301
21.4.4	SDA/SCL line control . . . . .	302
21.5	I2C low power modes . . . . .	303
21.6	I2C interrupts . . . . .	303
21.7	I2C registers . . . . .	305
21.7.1	Control register 1 (I2C_CR1) . . . . .	305
21.7.2	Control register 2 (I2C_CR2) . . . . .	306
21.7.3	Frequency register (I2C_FREQR) . . . . .	307
21.7.4	Own address register LSB (I2C_OARL) . . . . .	308
21.7.5	Own address register MSB (I2C_OARH) . . . . .	308
21.7.6	Data register (I2C_DR) . . . . .	309
21.7.7	Status register 1 (I2C_SR1) . . . . .	310
21.7.8	Status register 2 (I2C_SR2) . . . . .	312
21.7.9	Status register 3 (I2C_SR3) . . . . .	313
21.7.10	Interrupt register (I2C_ITR) . . . . .	314
21.7.11	Clock control register low (I2C_CCRL) . . . . .	315
21.7.12	Clock control register high (I2C_CCRH) . . . . .	316
21.7.13	TRISE register (I2C_TRISER) . . . . .	318
21.7.14	I2C register map and reset values . . . . .	319
<b>22</b>	<b>Universal asynchronous receiver transmitter (UART) . . . . .</b>	<b>320</b>
22.1	Introduction . . . . .	320
22.2	UART main features . . . . .	321
22.3	UART functional description . . . . .	322
22.3.1	UART character description . . . . .	327
22.3.2	Transmitter . . . . .	328
22.3.3	Receiver . . . . .	331
22.3.4	High precision baud rate generator . . . . .	336
22.3.5	Clock deviation tolerance of the UART receiver . . . . .	337
22.3.6	Parity control . . . . .	338
22.3.7	Multi-processor communication . . . . .	339
22.3.8	LIN (local interconnection network) mode . . . . .	340
22.3.9	UART synchronous communication . . . . .	341
22.3.10	Single wire half duplex communication . . . . .	343
22.3.11	Smartcard . . . . .	343

22.3.12	IrDA SIR ENDEC block . . . . .	345
22.4	LIN mode functional description . . . . .	348
22.4.1	Master mode . . . . .	348
22.4.2	Slave mode with automatic resynchronization disabled . . . . .	352
22.4.3	Slave mode with automatic resynchronization enabled . . . . .	355
22.4.4	LIN mode selection . . . . .	360
22.5	UART low power modes . . . . .	361
22.6	UART interrupts . . . . .	361
22.7	UART registers . . . . .	363
22.7.1	Status register (UART_SR) . . . . .	363
22.7.2	Data register (UART_DR) . . . . .	365
22.7.3	Baud rate register 1 (UART_BRR1) . . . . .	365
22.7.4	Baud rate register 2 (UART_BRR2) . . . . .	366
22.7.5	Control register 1 (UART_CR1) . . . . .	366
22.7.6	Control register 2 (UART_CR2) . . . . .	367
22.7.7	Control register 3 (UART_CR3) . . . . .	369
22.7.8	Control register 4 (UART_CR4) . . . . .	370
22.7.9	Control register 5 (UART_CR5) . . . . .	371
22.7.10	Control register 6 (UART_CR6) . . . . .	372
22.7.11	Guard time register (UART_GTR) . . . . .	373
22.7.12	Prescaler register (UART_PSCR) . . . . .	374
22.7.13	UART register map and reset values . . . . .	375
<b>23</b>	<b>Controller area network (beCAN) . . . . .</b>	<b>378</b>
23.1	Introduction . . . . .	378
23.2	beCAN main features . . . . .	378
23.3	beCAN general description . . . . .	379
23.3.1	CAN 2.0B active core . . . . .	379
23.3.2	Control, status and configuration registers . . . . .	379
23.3.3	Tx mailboxes . . . . .	380
23.3.4	Acceptance filters . . . . .	380
23.4	Operating modes . . . . .	381
23.4.1	Initialization mode . . . . .	381
23.4.2	Normal mode . . . . .	382
23.4.3	Sleep mode (low power) . . . . .	382
23.4.4	Time triggered communication mode . . . . .	382

---

23.5	Test modes . . . . .	383
23.5.1	Silent mode . . . . .	383
23.5.2	Loop back mode . . . . .	383
23.5.3	Loop back combined with silent mode . . . . .	384
23.6	Functional description . . . . .	384
23.6.1	Transmission handling . . . . .	384
23.6.2	Reception handling . . . . .	387
23.6.3	Identifier filtering . . . . .	388
23.6.4	Message storage . . . . .	394
23.6.5	Error management . . . . .	396
23.6.6	Bit timing . . . . .	397
23.7	Interrupts . . . . .	399
23.8	Register access protection . . . . .	400
23.9	Clock system . . . . .	400
23.10	beCAN low power modes . . . . .	400
23.11	beCAN registers . . . . .	401
23.11.1	CAN master control register (CAN_MCR) . . . . .	401
23.11.2	CAN master status register (CAN_MSR) . . . . .	402
23.11.3	CAN transmit status register (CAN_TSR) . . . . .	403
23.11.4	CAN transmit priority register (CAN_TPR) . . . . .	404
23.11.5	CAN receive FIFO register (CAN_RFR) . . . . .	406
23.11.6	CAN interrupt enable register (CAN_IER) . . . . .	407
23.11.7	CAN diagnostic register (CAN_DGR) . . . . .	408
23.11.8	CAN page select register (CAN_PSR) . . . . .	408
23.11.9	CAN error status register (CAN_ESR) . . . . .	409
23.11.10	CAN error interrupt enable register (CAN_EIER) . . . . .	410
23.11.11	CAN transmit error counter register (CAN_TECR) . . . . .	410
23.11.12	CAN receive error counter register (CAN_RECER) . . . . .	411
23.11.13	CAN bit timing register 1 (CAN_BTR1) . . . . .	411
23.11.14	CAN bit timing register 2 (CAN_BTR2) . . . . .	412
23.11.15	Mailbox registers . . . . .	413
23.11.16	CAN filter registers . . . . .	418
23.12	CAN register map . . . . .	424
23.12.1	Page mapping for CAN . . . . .	425
24	Analog/digital converter (ADC) . . . . .	428

24.1	Introduction	428
24.2	ADC main features	428
24.3	ADC extended features	428
24.4	ADC pins	431
24.5	ADC functional description	431
24.5.1	ADC on-off control	431
24.5.2	ADC clock	431
24.5.3	Channel selection	432
24.5.4	Conversion modes	432
24.5.5	Overrun flag	433
24.5.6	Analog watchdog	434
24.5.7	Conversion on external trigger	435
24.5.8	Analog zooming	435
24.5.9	Timing diagram	435
24.6	ADC low power modes	437
24.7	ADC interrupts	437
24.8	Data alignment	440
24.9	Reading the conversion result	440
24.10	Schmitt trigger disable registers	441
24.11	ADC registers	441
24.11.1	ADC data buffer register x high (ADC_DBxRH) (x=0..7 or 0..9 )	441
24.11.2	ADC data buffer register x low (ADC_DBxRL) (x=0..7 or 0..9 )	442
24.11.3	ADC control/status register (ADC_CSR)	443
24.11.4	ADC configuration register 1 (ADC_CR1)	444
24.11.5	ADC configuration register 2 (ADC_CR2)	445
24.11.6	ADC configuration register 3 (ADC_CR3)	446
24.11.7	ADC data register high (ADC_DRH)	447
24.11.8	ADC data register low (ADC_DRL)	447
24.11.9	ADC Schmitt trigger disable register high (ADC_TDRH)	448
24.11.10	ADC Schmitt trigger disable register low (ADC_TDRL)	448
24.11.11	ADC high threshold register high (ADC_HTRH)	449
24.11.12	ADC high threshold register low (ADC_HTRL)	449
24.11.13	ADC low threshold register high (ADC_LTRH)	450
24.11.14	ADC low threshold register low (ADC_LTDL)	450
24.11.15	ADC watchdog status register high (ADC_AWSRH)	451
24.11.16	ADC watchdog status register low (ADC_AWSRL)	451

---

24.11.17 ADC watchdog control register high (ADC_AWCRH) .....	452
24.11.18 ADC watchdog control register low (ADC_AWCRL) .....	452
24.12 ADC register map and reset values .....	453
<b>25 Revision history .....</b>	<b>455</b>

## List of tables

Table 1.	Interrupt levels . . . . .	26
Table 2.	CPU register map . . . . .	27
Table 3.	CFG_GCR register map . . . . .	28
Table 4.	List of abbreviations . . . . .	33
Table 5.	Block size . . . . .	49
Table 6.	Memory access versus programming method . . . . .	50
Table 7.	Flash register map and reset values . . . . .	56
Table 8.	Software priority levels . . . . .	60
Table 9.	Interrupt enabling/disabling inside an ISR . . . . .	60
Table 10.	Vector address map versus software priority bits . . . . .	65
Table 11.	Dedicated interrupt instruction set . . . . .	66
Table 12.	Interrupt register map . . . . .	72
Table 13.	RST register map . . . . .	77
Table 14.	Devices with 4 trimming bits . . . . .	82
Table 15.	Devices with 3 trimming bits . . . . .	82
Table 16.	CLK interrupt requests . . . . .	89
Table 17.	Peripheral clock gating bits . . . . .	95
Table 18.	Peripheral clock gating bits . . . . .	96
Table 19.	CLK register map and reset values . . . . .	100
Table 20.	Low power mode management . . . . .	102
Table 21.	I/O port configuration summary . . . . .	107
Table 22.	Effect of low power modes on GPIO ports . . . . .	108
Table 23.	Recommended and non-recommended configurations for analog input . . . . .	109
Table 24.	GPIO register map . . . . .	113
Table 25.	Time base calculation table . . . . .	116
Table 26.	AWU register map . . . . .	120
Table 27.	Beeper register map . . . . .	123
Table 28.	Watchdog timeout period (LSI clock frequency = 128 kHz) . . . . .	125
Table 29.	IWDG register map . . . . .	127
Table 30.	Window watchdog timing example . . . . .	131
Table 31.	Effect of low power modes on WWDG . . . . .	131
Table 32.	WWDG register map and reset values . . . . .	133
Table 33.	Timer characteristics . . . . .	134
Table 34.	Timer feature comparison . . . . .	135
Table 35.	Glossary of internal timer signals . . . . .	136
Table 36.	Explanation of indices 'i', 'n', and 'x' . . . . .	137
Table 37.	Counting direction versus encoder signals . . . . .	185
Table 38.	Output control for complementary OCi and OCiN channels with break feature . . . . .	207
Table 39.	TIM1 register map . . . . .	219
Table 40.	TIM2 register map . . . . .	247
Table 41.	TIM3 register map . . . . .	249
Table 42.	TIM5 register map . . . . .	250
Table 43.	TIM4 register map . . . . .	260
Table 44.	TIM6 register map . . . . .	261
Table 45.	SPI behavior in low power modes . . . . .	280
Table 46.	SPI interrupt requests . . . . .	280
Table 47.	SPI register map and reset values . . . . .	287
Table 48.	I2C interface behavior in low power modes . . . . .	303

---

Table 49.	I2C Interrupt requests . . . . .	303
Table 50.	I2C_CCR values for SCL frequency table (fMASTER = 10 MHz or 16 MHz) . . . . .	317
Table 51.	I2C register map . . . . .	319
Table 52.	UART configurations . . . . .	320
Table 53.	Noise detection from sampled data . . . . .	335
Table 54.	Baud rate programming and error calculation . . . . .	337
Table 55.	UART receiver tolerance when UART_DIV[3:0] is zero . . . . .	337
Table 56.	UART receiver's tolerance when UART_DIV[3:0] is different from zero . . . . .	338
Table 57.	Frame format . . . . .	338
Table 58.	LIN mode selection . . . . .	360
Table 59.	UART interface behavior in low power modes . . . . .	361
Table 60.	UART interrupt requests . . . . .	361
Table 61.	UART1 register map . . . . .	375
Table 62.	UART2 register map . . . . .	375
Table 63.	UART3 register map . . . . .	376
Table 64.	UART4 register map . . . . .	377
Table 65.	Example of filter numbering . . . . .	392
Table 66.	Transmit mailbox mapping . . . . .	394
Table 67.	Receive mailbox mapping . . . . .	395
Table 68.	beCAN behavior in low power modes . . . . .	400
Table 69.	beCAN control and status page - register map and reset values . . . . .	426
Table 70.	beCAN mailbox pages - register map and reset values . . . . .	426
Table 71.	beCAN filter configuration page - register map and reset values . . . . .	427
Table 72.	ADC pins . . . . .	431
Table 73.	Low power modes . . . . .	437
Table 74.	ADC Interrupts in single and non-buffered continuous mode (ADC1 and ADC2) . . . . .	437
Table 75.	ADC interrupts in buffered continuous mode (ADC1) . . . . .	438
Table 76.	ADC interrupts in scan mode (ADC1) . . . . .	439
Table 77.	ADC1 register map and reset values . . . . .	453
Table 78.	ADC2 register map and reset values . . . . .	454
Table 79.	Document revision history . . . . .	455

## List of figures

Figure 1.	Programming model . . . . .	24
Figure 2.	Stacking order. . . . .	25
Figure 3.	Memory map. . . . .	30
Figure 4.	Default stack model . . . . .	31
Figure 5.	Customized stack model. . . . .	32
Figure 6.	Flash memory and data EEPROM organization on low density STM8S and STM8AF . . . . .	38
Figure 7.	Flash memory and data EEPROM organization on medium density STM8S and STM8AF39	
Figure 8.	Flash memory and data EEPROM organization high density STM8S and STM8AF . . . . .	40
Figure 9.	UBC area size definition on low density STM8S devices . . . . .	41
Figure 10.	UBC area size definition on medium density STM8S and STM8AF with up to 32 Kbytes of Flash program memory . . . . .	42
Figure 11.	UBC area size definition on high density STM8S and STM8AF with up to 128 Kbytes of Flash program memory . . . . .	43
Figure 12.	SWIM pin connection . . . . .	57
Figure 13.	Interrupt processing flowchart . . . . .	60
Figure 14.	Priority decision process . . . . .	61
Figure 15.	Concurrent interrupt management . . . . .	64
Figure 16.	Nested interrupt management . . . . .	66
Figure 17.	Power supply overview . . . . .	73
Figure 18.	Reset circuit . . . . .	74
Figure 19.	VDD/VDDIO voltage detection: POR/BOR threshold . . . . .	75
Figure 20.	Clock tree . . . . .	79
Figure 21.	HSE clock sources . . . . .	80
Figure 22.	Clock switching flowchart (automatic mode example) . . . . .	85
Figure 23.	Clock switching flowchart (manual mode example) . . . . .	86
Figure 24.	GPIO block diagram . . . . .	106
Figure 25.	AWU block diagram . . . . .	114
Figure 26.	Beep block diagram . . . . .	121
Figure 27.	Independent watchdog (IWDG) block diagram . . . . .	124
Figure 28.	Watchdog block diagram . . . . .	129
Figure 29.	Approximate timeout duration . . . . .	130
Figure 30.	Window watchdog timing diagram . . . . .	131
Figure 31.	TIM1 general block diagram . . . . .	140
Figure 32.	Time base unit . . . . .	141
Figure 33.	16-bit read sequence for the counter (TIM1_CNTR) . . . . .	142
Figure 34.	Counter in up-counting mode . . . . .	143
Figure 35.	Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2 . . . . .	144
Figure 36.	Counter update event when ARPE = 1 (TIM1_ARR preloaded) . . . . .	144
Figure 37.	Counter in down-counting mode . . . . .	145
Figure 38.	Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2 . . . . .	146
Figure 39.	Counter update when ARPE = 1 (ARR preloaded), with prescaler = 1 . . . . .	146
Figure 40.	Counter in center-aligned mode . . . . .	147
Figure 41.	Counter timing diagram, $f_{CK\_CNT} = f_{CK\_PSC}$ , TIM1_ARR = 06h, ARPE = 1 . . . . .	148
Figure 42.	Update rate examples depending on mode and TIM1_RCR register settings . . . . .	150
Figure 43.	Clock/trigger controller block diagram . . . . .	151
Figure 44.	Control circuit in normal mode, $f_{CK\_PSC} = f_{MASTER}$ . . . . .	152
Figure 45.	TI2 external clock connection example . . . . .	152
Figure 46.	Control circuit in external clock mode 1 . . . . .	153

---

Figure 47.	External trigger input block diagram .....	154
Figure 48.	Control circuit in external clock mode 2 .....	154
Figure 49.	Control circuit in trigger mode .....	155
Figure 50.	Control circuit in trigger reset mode .....	156
Figure 51.	Control circuit in trigger gated mode .....	157
Figure 52.	Control circuit in external clock mode 2 + trigger mode .....	158
Figure 53.	Timer chaining system implementation example .....	160
Figure 54.	Trigger/master mode selection blocks .....	161
Figure 55.	Master/slave timer example .....	161
Figure 56.	Gating timer B with OC1REF of timer A .....	162
Figure 57.	Gating timer B with the counter enable signal of timer A (CNT_EN) .....	163
Figure 58.	Triggering timer B with the UEV of timer A (TIMERA-UEV) .....	164
Figure 59.	Triggering timer B with counter enable CNT_EN of timer A .....	165
Figure 60.	Triggering Timer A and B with Timer A TI1 input .....	166
Figure 61.	Capture/compare channel 1 main circuit .....	166
Figure 62.	16-bit read sequence for the TIM1_CCR <sub>i</sub> register in capture mode .....	167
Figure 63.	Channel input stage block diagram .....	168
Figure 64.	Input stage of TIM 1 channel 1 .....	168
Figure 65.	PWM input signal measurement .....	170
Figure 66.	PWM input signal measurement example .....	171
Figure 67.	Channel output stage block diagram .....	171
Figure 68.	Detailed output stage of channel with complementary output (channel 1) .....	172
Figure 69.	Output compare mode, toggle on OC1 .....	173
Figure 70.	Edge-aligned counting mode PWM mode 1 waveforms (ARR = 8) .....	175
Figure 71.	Center-aligned PWM waveforms (ARR = 8) .....	176
Figure 72.	Example of one-pulse mode .....	177
Figure 73.	Complementary output with deadtime insertion .....	179
Figure 74.	Deadtime waveforms with a delay greater than the negative pulse .....	179
Figure 75.	Deadtime waveforms with a delay greater than the positive pulse .....	179
Figure 76.	Six-step generation, COM example (OSSR = 1) .....	181
Figure 77.	Behavior of outputs in response to a break (channel without complementary output) .....	182
Figure 78.	Behavior of outputs in response to a break (TIM1 complementary outputs) .....	183
Figure 79.	ETR activation .....	184
Figure 80.	Example of counter operation in encoder interface mode .....	186
Figure 81.	Example of encoder interface mode with IC1 polarity inverted .....	186
Figure 82.	TIM2/TIM3 block diagram .....	223
Figure 83.	TIM5 block diagram .....	224
Figure 84.	Time base unit .....	224
Figure 85.	Input stage block diagram .....	226
Figure 86.	Input stage of TIM 2 channel 1 .....	226
Figure 87.	Output stage .....	227
Figure 88.	Output stage of channel 1 .....	227
Figure 89.	TIM4 block diagram .....	253
Figure 90.	TIM6 block diagram .....	253
Figure 91.	SPI block diagram .....	263
Figure 92.	Single master/ single slave application .....	264
Figure 93.	Data clock timing diagram .....	266
Figure 94.	TXE/RXNE/BSY behavior in full duplex mode (RXONLY = 0). Case of continuous transfers .....	271
Figure 95.	TXE/RXNE/BSY behavior in slave / full duplex mode (BDM = 0, RXONLY = 0). Case of continuous transfers .....	271
Figure 96.	TXE/BSY in master transmit-only mode .....	

---

(BDM = 0 and RXONLY = 0). Case of continuous transfers.....	272
Figure 97. TXE/BSY in slave transmit-only mode (BDM = 0 and RXONLY = 0).	
Case of continuous transfers .....	273
Figure 98. RXNE behavior in receive-only mode (BDM = 0 and RXONLY = 1).	
Case of continuous transfers .....	274
Figure 99. TXE/BSY behavior when transmitting (BDM = 0 and RXONLY = 0).	
Case of discontinuous transfers .....	275
Figure 100. I2Cbus protocol.....	289
Figure 101. I2C block diagram .....	290
Figure 102. Transfer sequence diagram for slave transmitter.....	292
Figure 103. Transfer sequence diagram for slave receiver .....	293
Figure 104. Transfer sequence diagram for master transmitter.....	296
Figure 105. Method 1: transfer sequence diagram for master receiver.....	297
Figure 106. Method 2: transfer sequence diagram for master receiver when N >2.....	298
Figure 107. Method 2: transfer sequence diagram for master receiver when N=2 .....	300
Figure 108. Method 2: transfer sequence diagram for master receiver when N=1 .....	300
Figure 109. I2C interrupt mapping diagram.....	304
Figure 110. UART1 block diagram.....	323
Figure 111. UART2 block diagram.....	324
Figure 112. UART3 block diagram.....	325
Figure 113. UART4 block diagram.....	326
Figure 114. Word length programming .....	327
Figure 115. Configurable stop bits .....	329
Figure 116. TC/TXE behavior when transmitting .....	330
Figure 117. Start bit detection .....	332
Figure 118. Data sampling for noise detection .....	334
Figure 119. How to code UART_DIV in the BRR registers .....	336
Figure 120. Mute mode using idle line detection .....	339
Figure 121. Mute mode using Address mark detection .....	340
Figure 122. UART example of synchronous transmission.....	342
Figure 123. UART data clock timing diagram (M=0) .....	342
Figure 124. UART data clock timing diagram (M=1) .....	342
Figure 125. RX data setup/hold time .....	343
Figure 126. ISO 7816-3 asynchronous protocol .....	344
Figure 127. Parity error detection using 1.5 stop bits .....	345
Figure 128. IrDA SIR ENDEC- block diagram .....	347
Figure 129. IrDA data modulation (3/16) - normal mode .....	347
Figure 130. Break detection in LIN mode (11-bit break length - LBDL bit is set).....	350
Figure 131. Break detection in LIN mode vs framing error detection.....	351
Figure 132. LIN identifier field parity bits .....	353
Figure 133. LIN identifier field parity check .....	353
Figure 134. LIN header reception time-out .....	354
Figure 135. LIN synch field measurement .....	356
Figure 136. UARTDIV read / write operations when LDUM = 0.....	356
Figure 137. UARTDIV read / write operations when LDUM = 1.....	357
Figure 138. Bit sampling in reception mode.....	360
Figure 139. UART interrupt mapping diagram .....	362
Figure 140. CAN network topology .....	379
Figure 141. beCAN block diagram.....	380
Figure 142. beCAN operating modes .....	381
Figure 143. beCAN in silent mode .....	383
Figure 144. beCAN in loop back mode .....	383

---

Figure 145. beCAN in combined mode . . . . .	384
Figure 146. Transmit mailbox states . . . . .	386
Figure 147. Receive FIFO states . . . . .	387
Figure 148. 32-bit filter bank configuration (FSCx bits = 0b11 in CAN_FCRx register) . . . . .	390
Figure 149. 16-bit filter bank configuration (FSCx bits = 0b10 in CAN_FCRx register) . . . . .	390
Figure 150. 16/8-bit filter bank configuration (FSCx bits = 0b01 in CAN_FCRx register) . . . . .	391
Figure 151. 8-bit filter bank configuration (FSCx bits = 0b00 in CAN_FCRx register) . . . . .	391
Figure 152. Filter banks configured as in the example in <i>Table 65</i> . . . . .	393
Figure 153. CAN error state diagram . . . . .	396
Figure 154. Bit timing . . . . .	397
Figure 155. CAN frames . . . . .	398
Figure 156. Event flags and interrupt generation . . . . .	399
Figure 157. CAN register mapping . . . . .	424
Figure 158. CAN page mapping . . . . .	425
Figure 159. ADC1 block diagram . . . . .	429
Figure 160. ADC2 block diagram . . . . .	430
Figure 161. Analog watchdog guarded area . . . . .	434
Figure 162. Timing diagram in single mode (CONT = 0) . . . . .	436
Figure 163. Timing diagram in continuous mode (CONT = 1) . . . . .	436
Figure 164. Right alignment of data . . . . .	440
Figure 165. Left alignment of data . . . . .	440

# 1 Central processing unit (CPU)

## 1.1 CPU 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).

## 1.2 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.

### 1.2.1 Description of CPU registers

#### Accumulator (A)

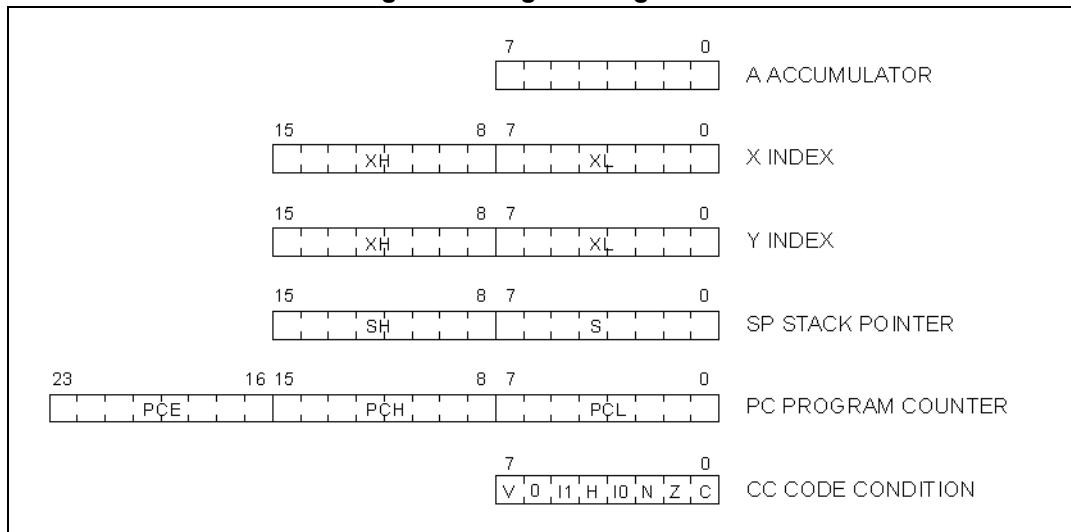
The accumulator is an 8-bit general purpose register used to hold operands and the results of the arithmetic and logic calculations as well as data manipulations.

#### 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.

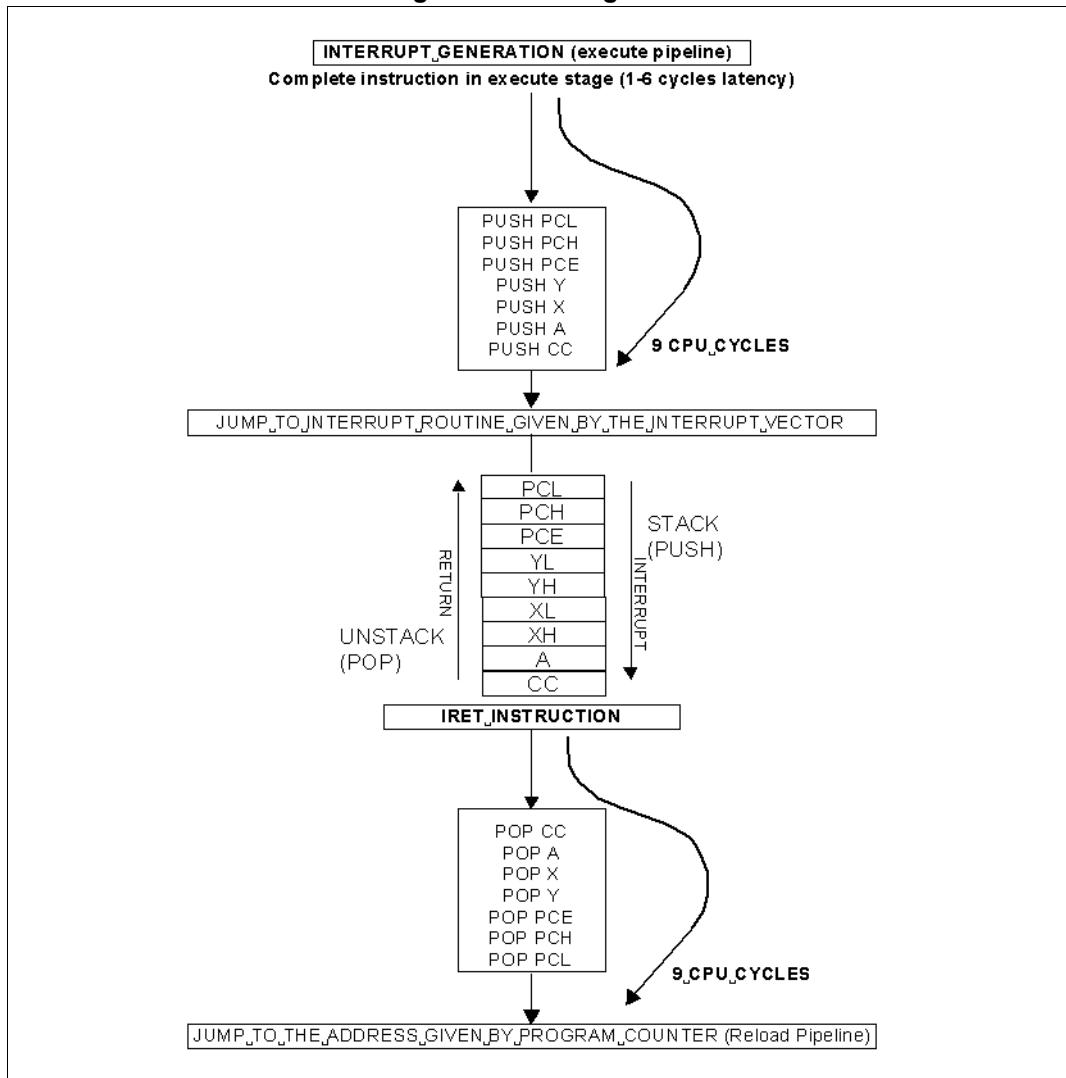
The stack pointer can be initialized by the startup function provided with the C compiler. For applications written in C language, the initialization is then performed according to the address specified in the linker file for C users. If you use your own linker file or startup file, make sure the stack 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 1](#). 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 1. Interrupt levels**

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

$$\begin{array}{r}
 & \text{C} & \text{7} & & & & & \text{0} \\
 & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & \boxed{1} \\
 + & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{0} \\
 = & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1}
 \end{array}$$

### 1.2.2 STM8 CPU register map

The CPU registers are mapped in the STM8 address space as shown in [Table 2](#). These registers can only be accessed by the debug module but not by memory access instructions executed in the core.

**Table 2. 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

## 1.3 Global configuration register (CFG\_GCR)

### 1.3.1 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 6.4: Activation level/low power mode control on page 63](#).

### 1.3.2 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.

### 1.3.3 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.

### 1.3.4 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 3. 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

## 2 Boot ROM

The internal 2 Kbyte boot ROM (available in some devices) contains the bootloader code. Its main tasks are to download the application program to the internal Flash/EEPROM through the SPI, CAN, or UART interface, and to program the code, data, option bytes and interrupt vectors in internal Flash/EEPROM.

To perform bootloading in LIN mode, a different bootloader communication protocol is implemented on UART2/UART3 and UART1.

The boot loader starts executing after reset. Refer to the STM8 bootloader user manual (UM0560) for more details.

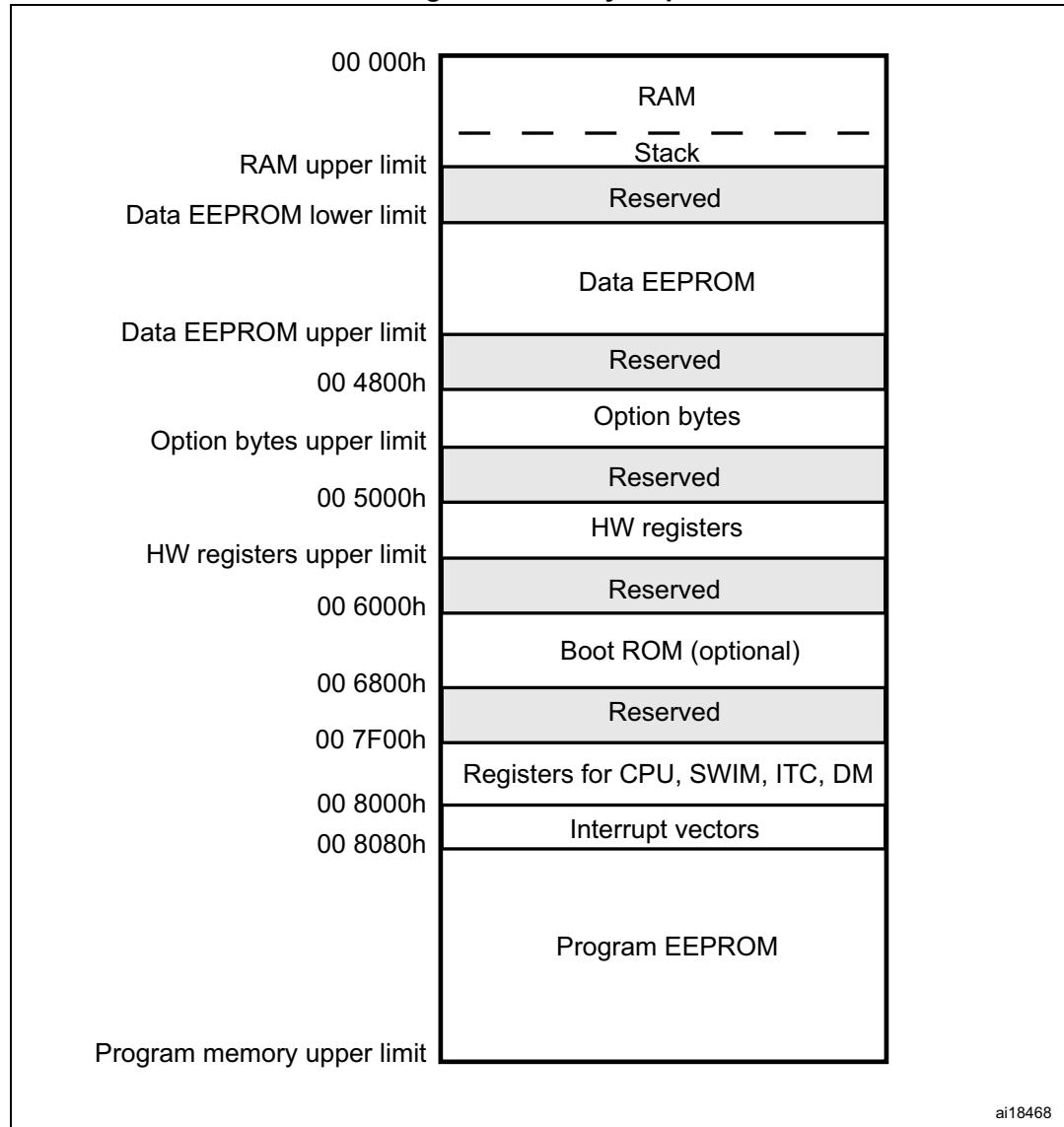
## 3 Memory and register map

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.

### 3.1 Memory layout

#### 3.1.1 Memory map

Figure 3. Memory map



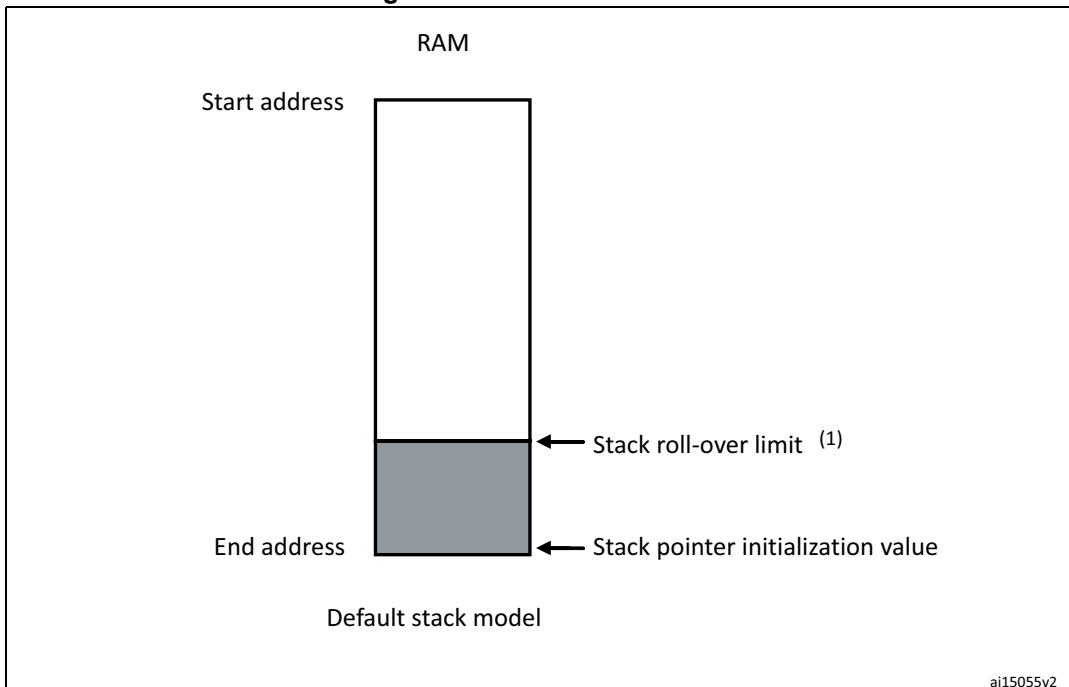
The RAM upper limit, data EEPROM upper and lower limit, Option Byte upper limit, hardware (HW) registers upper limit, and the program memory upper limit are specific to the device configuration. Please refer to the datasheets for quantitative information.

### 3.1.2 Stack handling

#### Default stack model

The stack of the STM8S and STM8AF microcontrollers is implemented in the user RAM area. The default stack model is shown in [Figure 4](#).

**Figure 4. Default stack model**



1. The stack roll-over limit is not implemented on all devices. Refer to the datasheets for detailed information.

#### Stack pointer initialization value

This is the default value of the stack pointer. The user must take care to initialize this pointer. Correct loading of this pointer is usually performed by the initialization code generated by the development tools (linker file). In the default stack model this pointer is initialized to the RAM end address.

#### Stack roll-over limit

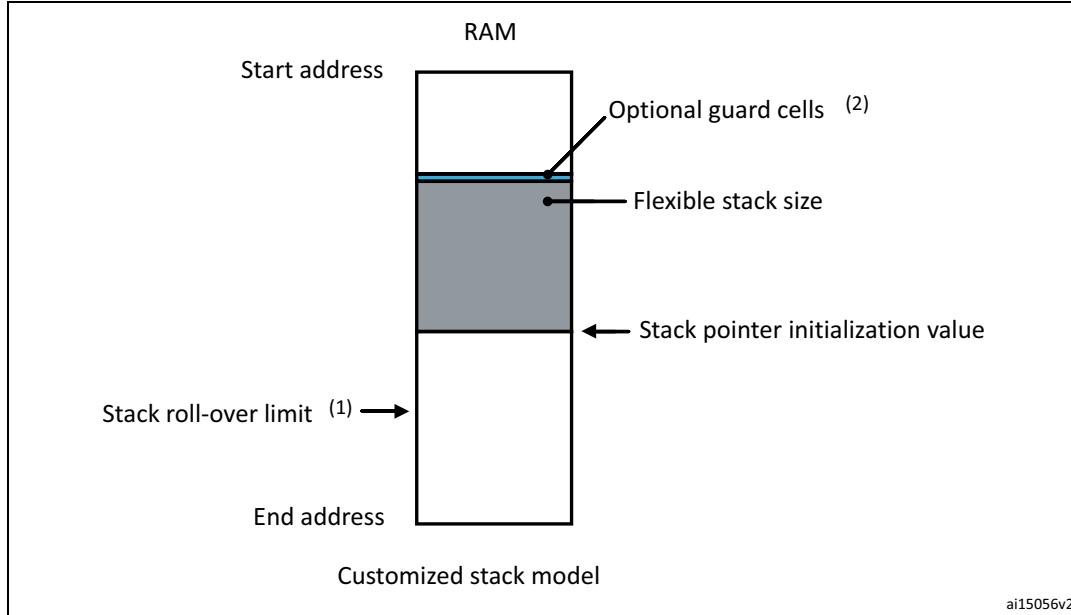
In some devices, a stack roll-over limit is implemented at a fixed address. If the stack pointer is decreased below the stack roll-over limit, using a push operation or during context saving for subroutines or interrupt routines, it is reset to the RAM end address. The stack pointer does not roll over if stack pointer arithmetic is used.

Such behavior of the stack pointer is of particular importance when developing software on a device with a different memory configuration than the target device.

### Customized stack model

STM8S and STM8AF stack pointer handling allows a customized stack model to be implemented. This permits a flexible stack size without restrictions due to the stack roll-over limit. Implementing the customized stack also benefits portability of the software on products with different memory configurations. [Figure 5](#) shows the customized stack model.

**Figure 5. Customized stack model**



ai15056v2

1. The stack roll-over limit is not implemented on all devices.
2. The guard cells are RAM locations that have to be continuously polled by the application program to detect whether a stack overflow has taken place.

In this stack model, the initial stack pointer must be placed beyond the stack roll-over limit. Consequently, the growing stack never reaches the stack roll-over limit. It is clear that in this implementation the stack size is not limited by the roll-over mechanism. Nevertheless, the user has to define the stack position and stack size in the link file, and he has to ensure that the stack pointer does not exceed the defined stack area (stack overflow or under-run).

The RAM locations above and below the customized stack can be regularly used as RAM to store variables or other information.

Guard cells can be implemented at the lower end of the stack to detect if the stack pointer exceeds the defined limit. These cells are standard RAM locations, initialized with fixed values that the stack overwrites if an overflow occurs. The user software can regularly poll these cells, detect the overflow condition, and put the application in a fail safe state.

During the software validation phase hardware breakpoints can be set at both limits of the stack to validate that neither a stack overflow nor an under-run happens.

## 3.2 Register description abbreviations

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

**Table 4. List of abbreviations**

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

## 4 Flash program memory and data EEPROM

### 4.1 Flash and EEPROM 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, or configure specific low power modes. The application can also program the device option bytes.

### 4.2 Flash and EEPROM 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.  
A dedicated option byte can be used to configure, by increments of one page, the size of the user boot code.

### 4.3 Main Flash memory features

- STM8S and STM8AF EEPROM is divided into two memory areas
  - Up to 128 Kbytes of Flash program memory. The density differs according to the device. Refer to [Section 4.4: Memory organization](#) for details
  - Up to 2 Kbytes of data EEPROM including option bytes. Data EEPROM density differs according to the device. Refer to [Section 4.4: Memory organization](#) for details.
- 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.
- Read-while-write capability (RWW). This feature is not available on all devices. Refer to the datasheets for details
- 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).
- Memory state configurable to operating or power-down ( $I_{DDQ}$ ) in Halt and Active-halt modes

## 4.4 Memory organization

### 4.4.1 STM8S and STM8AF memory organization

STM8S and STM8AF EEPROM is organized in 32-bit words (4 bytes per word).

The memory organization differs according to the devices:

- Low density STM8S and STM8AF devices
  - 8 Kbytes of Flash program memory organized in 128 pages or blocks of 64 bytes each. The Flash program memory is divided into 2 areas, the user boot code area (UBC), which size can be configured by option byte, and the main program memory area. The Flash program memory is mapped in the upper part of the STM8S addressing space and includes the reset and interrupt vectors.
  - Up to 640 bytes of data EEPROM (DATA) organized in pages or blocks of 64 bytes each. One block (64 bytes) contains the option bytes of which 11 are used to configure the device hardware features. The options bytes can be programmed in user, IAP and ICP/SWIM modes.
- Medium density STM8S devices
  - From 16 to 32 Kbytes of Flash program memory organized in up to 64 pages of 4 blocks of 128 bytes each. The Flash program memory is divided into 2 areas, the user boot code area (UBC), which size can be configured by option byte, and the main program memory area. The Flash program memory is mapped in the upper part of the STM8S addressing space and includes the reset and interrupt vectors.
  - Up to 1 Kbyte of data EEPROM (DATA) organized in up to 2 pages of 4 blocks of 128 bytes each. One block (128 bytes) contains the option bytes of which 13 are used to configure the device hardware features. The options bytes can be programmed in user, IAP and ICP/SWIM modes.
- Medium density STM8AF devices
  - From 16 to 32 Kbytes of Flash program memory organized in up to 64 pages of 4 blocks of 128 bytes each. The Flash program memory is divided into 2 areas, the user boot code area (UBC), which size can be configured by option byte, and the main program memory area. The Flash program memory is mapped in the upper part of the STM8AF addressing space and includes the reset and interrupt vectors.
  - Up to 1 Kbyte of data EEPROM (DATA) organized in up to 2 pages of 4 blocks of 128 bytes each. One block (128 bytes) contains the option bytes of which 13 are used to configure the device hardware features. The options bytes can be programmed in user, IAP and ICP/SWIM modes.
- High density STM8S devices
  - From 32 to 128 Kbytes of Flash program memory organized in up to 256 pages of 4 blocks of 128 bytes each. The Flash program memory is divided into 2 areas, the user boot code area (UBC), which size can be configured by option byte, and the main program memory area. The Flash program memory is mapped in the upper part of the STM8S addressing space and includes the reset and interrupt vectors.
  - Up to 2 Kbytes of data EEPROM (DATA) organized in up to 4 pages of 4 blocks of 128 bytes each. The size of the DATA area is fixed for a given microcontroller. One block (128 bytes) contains the option bytes of which 15 are used to configure the

device hardware features. The options bytes can be programmed in user, IAP and ICP/SWIM modes.

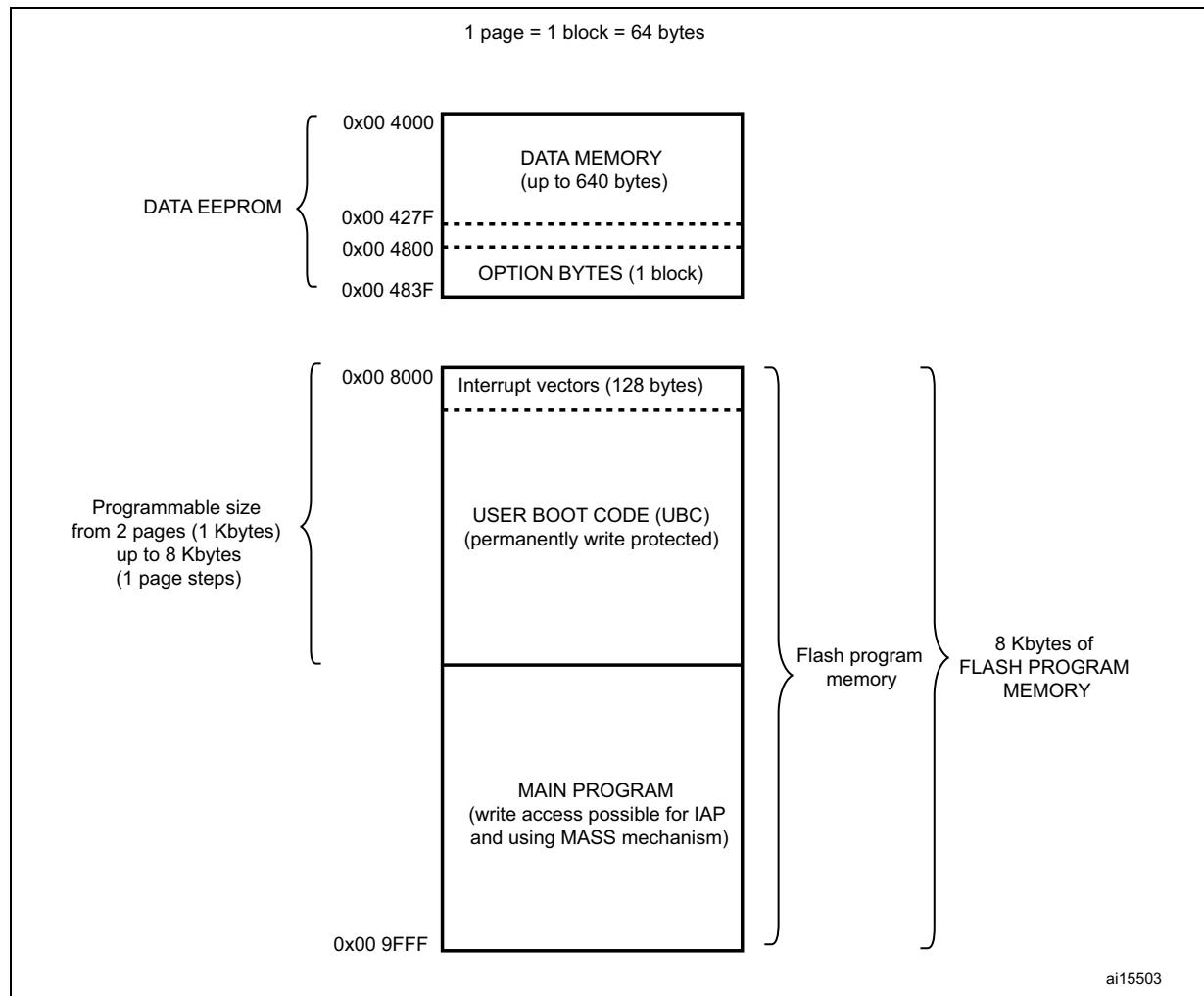
- High density STM8AF devices
  - From 32 to 128 Kbytes of Flash program memory organized in up to 256 pages of 4 blocks of 128 bytes each. The Flash program memory is divided into 2 areas, the user boot code area (UBC), which size can be configured by option byte, and the main program memory area. The Flash program memory is mapped in the upper part of the STM8AF addressing space and includes the reset and interrupt vectors.
  - Up to 2 Kbytes of data EEPROM (DATA) organized in up to 4 pages of 4 blocks of 128 bytes each. The size of the DATA area is fixed for a given microcontroller. One block (128 bytes) contains the option bytes of which 15 are used to configure the device hardware features. The options bytes can be programmed in user, IAP and ICP/SWIM modes.

The page defines the granularity of the user boot code area as described in [Section 4.4.3: User boot area \(UBC\)](#).

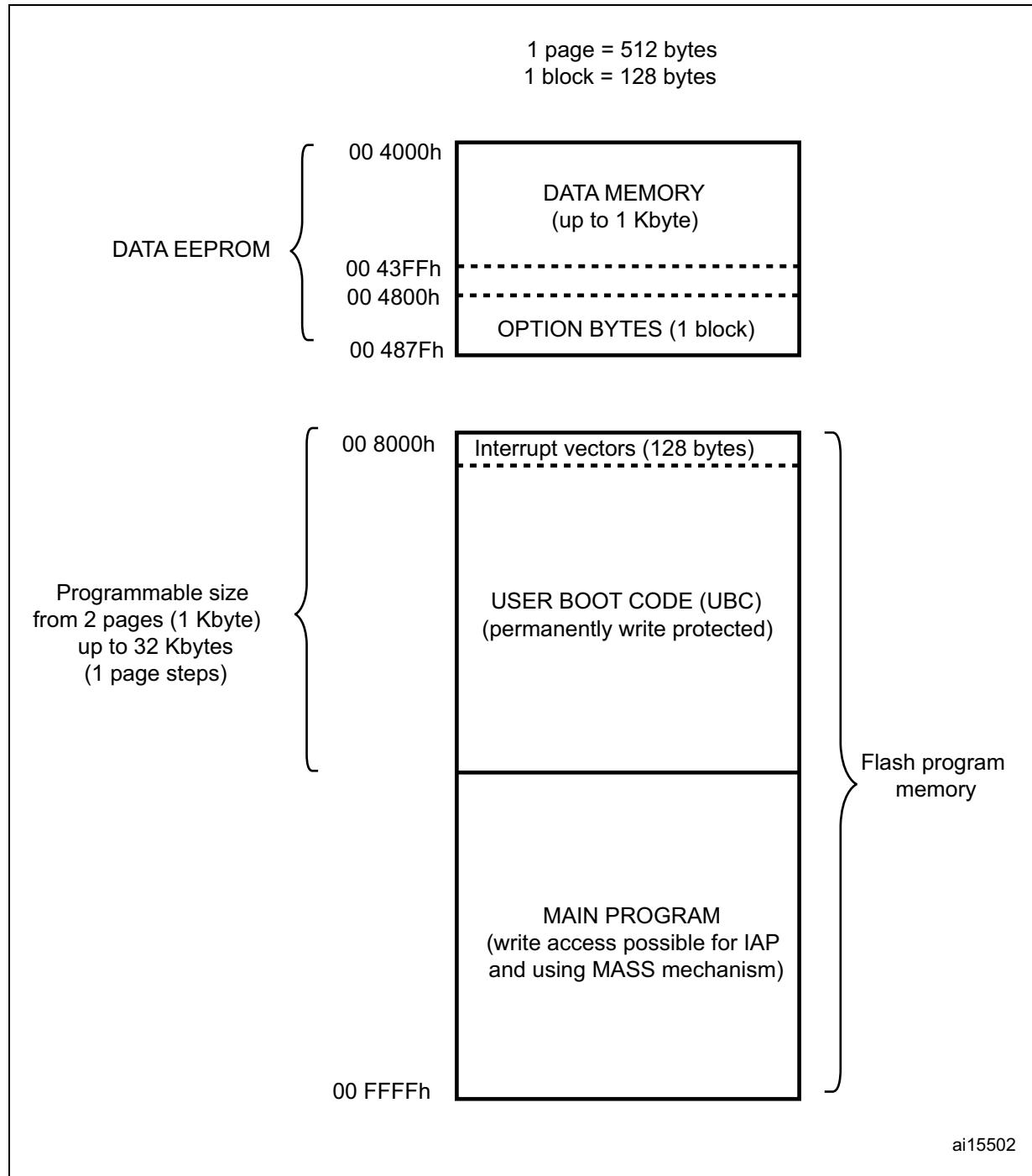
[Figure 6](#), [Figure 7](#), and [Figure 8](#) show the Flash memory and data EEPROM organization for STM8S and STM8AF devices. Refer to the STM8S and STM8AF programming manual (PM0051) for more information.

**Note:** *The EEPROM access time allows the device to run up to 16 MHz. For clock frequencies above 16 MHz, Flash/data EEPROM access must be configured for 1 wait state. This is enabled by the device option byte (refer to the option bytes section of the STM8S and STM8AF datasheets).*

**Figure 6. Flash memory and data EEPROM organization on low density STM8S and STM8AF**

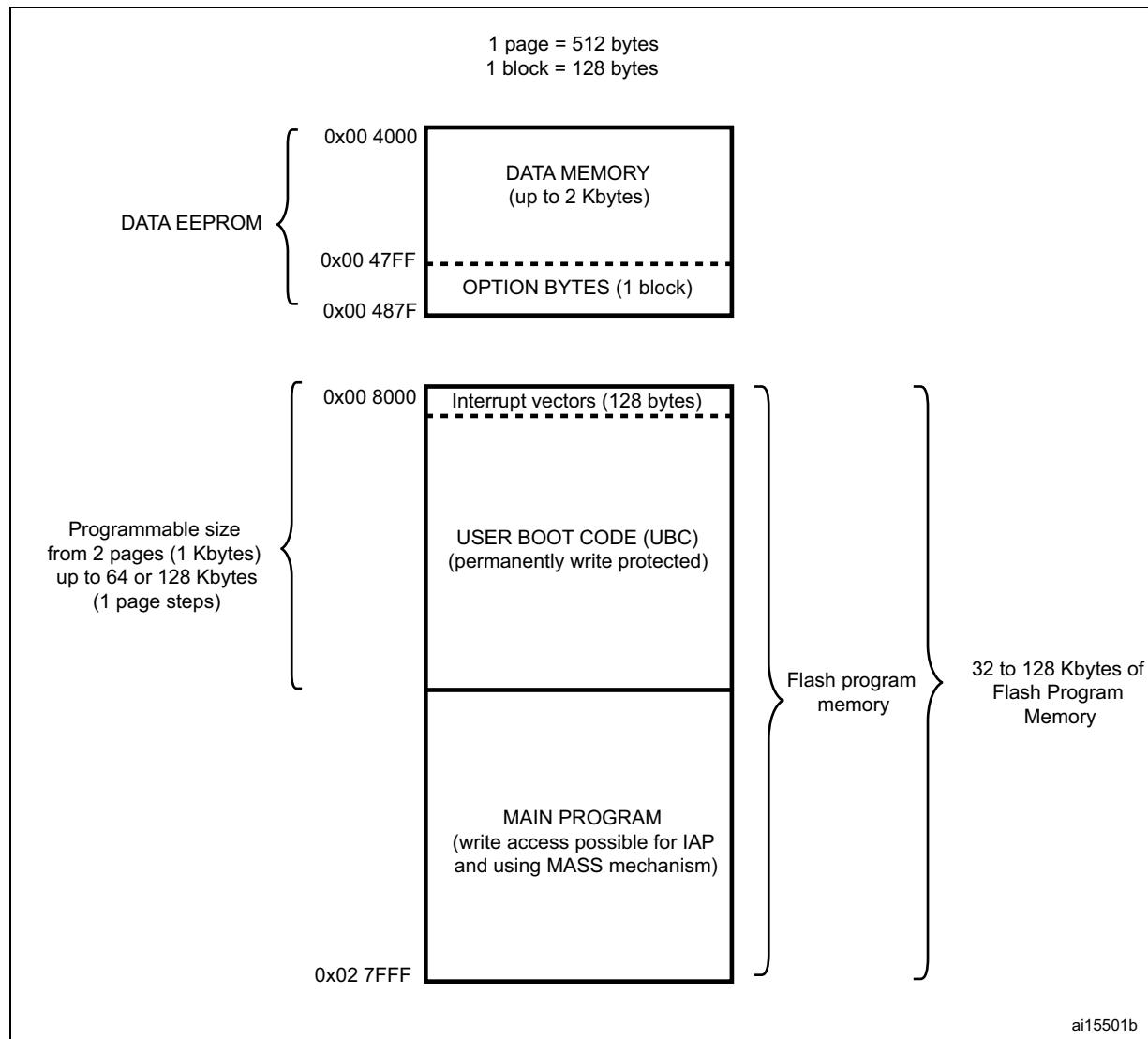


**Figure 7. Flash memory and data EEPROM organization on medium density STM8S and STM8AF**



1. The memory mapping is given for the STM8AF devices featuring 32 Kbytes of Flash program memory and 1 Kbytes of SRAM.

**Figure 8. Flash memory and data EEPROM organization high density STM8S and STM8AF**



#### 4.4.2 Memory access/ wait state configuration

The Flash/ data EEPROM access time allows the device to run at up to 16 MHz without wait states.

When using the high-speed external clock (HSE) at higher frequencies up to 24 MHz, one wait state is necessary. In this case the device option byte should be programmed to insert this wait state. Refer to the datasheet option byte section.

#### 4.4.3 User boot area (UBC)

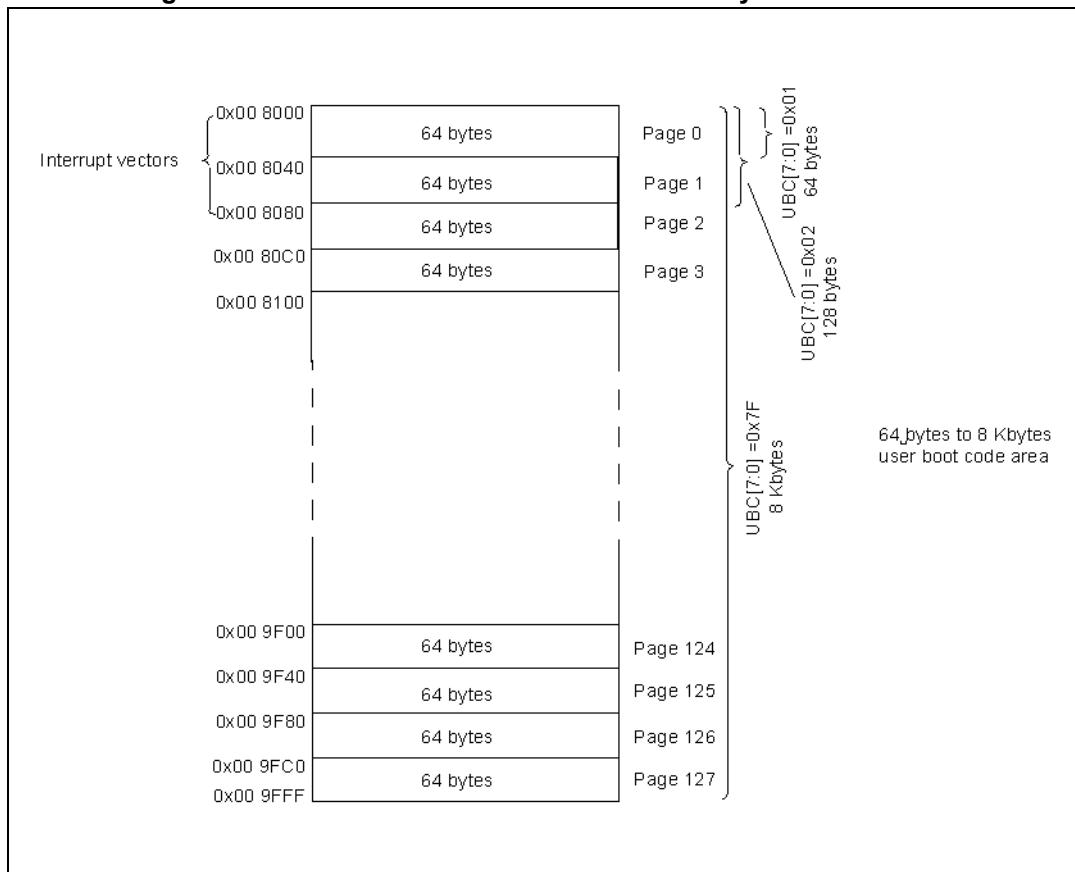
The user boot area (UBC) contains the reset and the interrupt vectors. It can be used to store the IAP and communication routines. The UBC area has a second level of protection to prevent unintentional erasing or modification during IAP programming. This means that it is always write protected and the 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.

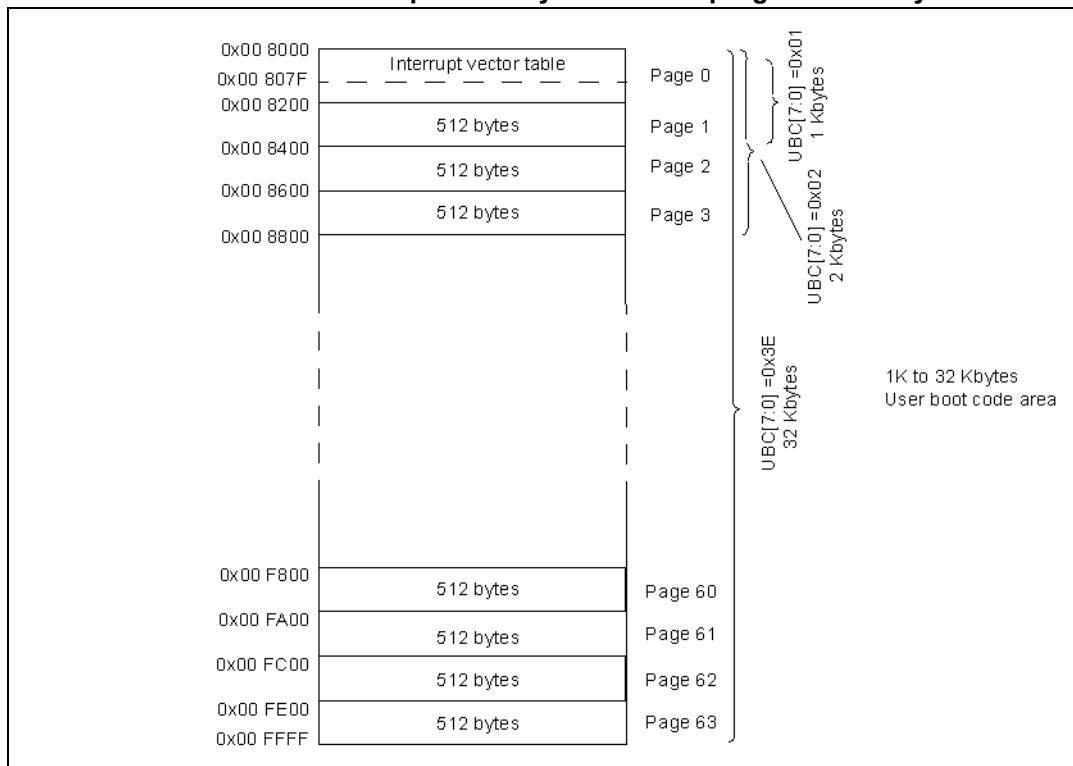
Refer to [Figure 9](#), [Figure 10](#), and [Figure 11](#) for a description of the UBC area memory mapping and to the option byte section in the datasheets for more details on the UBC option byte.

**Figure 9. UBC area size definition on low density STM8S devices**



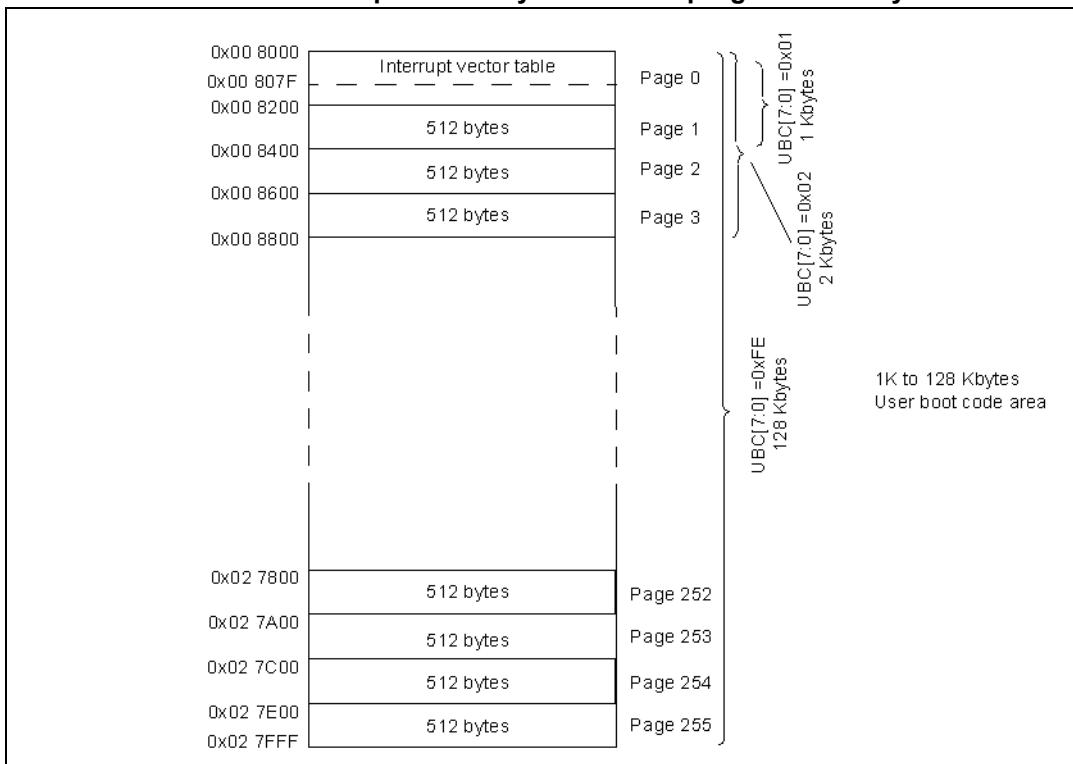
1. N (number of protected pages) = UBC[7:0].
2. UBC[7:0] = 0x00 means no user boot code area is defined. Refer to the datasheets for the description of the UBC option byte.
3. The first 2 pages (128 bytes) contain the interrupt vectors.

**Figure 10. UBC area size definition on medium density STM8S and STM8AF with up to 32 Kbytes of Flash program memory**



1. N (number of protected pages) = UBC[7:0] + 2 for UBC[7:0] > 1.
2. UBC[7:0] = 0x00 means no user boot code area is defined. Refer to the datasheets for the description of the UBC option byte.
3. The first 2 pages (1 Kbytes) contain the 128 bytes of interrupt vectors (32 IT vectors).

**Figure 11. UBC area size definition on high density STM8S and STM8AF with up to 128 Kbytes of Flash program memory**



1. UBC[7:0] = 0x00 means no user boot code area is defined. Refer to the datasheets for the description of the UBC option byte.
2. The first 2 pages (1 Kbytes) contain the interrupt vectors, of which only 128 bytes (32 IT vectors) are used.

#### 4.4.4 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](#)).

Refer to [Section 4.4: Memory organization](#) for the size of the DATA area.

#### 4.4.5 Main program area

The main program is the part of the Flash program memory which is used to store the application code (see [Figure 6](#), [Figure 7](#) and [Figure 8](#)).

#### 4.4.6 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 both in ICP/SWIM and in IAP mode, with OPT bit of the FLASH\_CR2 register set to 1 and the NOPT bit of the FLASH\_NCR2 register set to 0 (see [Section 4.8.2: Flash control register 2 \(FLASH\\_CR2\)](#) and [Section 4.8.3: Flash complementary control register 2 \(FLASH\\_NCR2\)](#)).

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. Furthermore, on medium and high density STM8S and STM8AF, the debug module (DM) cannot start code execution by the CPU when the readout protection is active, and the CPU is stalled.

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.

#### Removing the readout protection

The readout protection can be disabled on the program memory, UBC 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.

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 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 4.4.3: 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 4.8.6: 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 4.8.8: 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 4.8.9: 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 4.8.8: 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.

#### **4.5.3**

#### **Enabling write access to option bytes**

The procedure for enabling write access to the option byte area is the same as the one used for data EEPROM. However, the OPT bit in the [Flash control register 2 \(FLASH\\_CR2\)](#) must

be set, and the corresponding NOPT bit in the *Flash complementary control register 2 (FLASH\_NCR2)* must be cleared to enable write access to the option bytes.

## 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 \(MASS\)](#).

### 4.6.1 Read-while-write (RWW)

The RWW feature allows write operations to be performed on data EEPROM while reading and executing the program memory. Execution time is therefore optimized. The opposite operation is not allowed: Data memory cannot be read while writing to the program memory.

This RWW feature is always enabled and can be used at any time. Any access to Flash control registers FLASH\_CR1 and FLASH\_CR2 while writing to the memory stalls the CPU, making RWW unavailable.

*Note:* *The RWW feature is not available on all devices. Refer to the datasheets for addition information.*

### 4.6.2 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.

- In the main program memory:  
The application stops for the duration of the byte program operation.
- In DATA area:
  - Devices with RWW capability: Program execution does not stop, and the byte program operation is performed using the read-while-write (RWW) capability in IAP mode.
  - Devices without RWW capability: 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.*

### 4.6.3 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. On some devices, the read-while-write (RWW) capability is also available when a word programming operation is performed on the data EEPROM. Refer to the datasheets for additional information.

- In the main program memory:  
The application stops for the duration of the byte program operation.
- In DATA area
  - Devices with RWW capability: Program execution does not stop, and the byte program operation is performed using the read-while-write (RWW) capability in IAP mode.
  - Devices without RWW capability: The application stops for the duration of the byte program operation.

To program a word, the WPRG/NWPRG bits in the FLASH\_CR2 and FLASH\_NCR2 registers must be previously set/cleared to enable word programming mode (see [Section 4.8.2: Flash control register 2 \(FLASH\\_CR2\)](#) and [Section 4.8.2: Flash control register 2 \(FLASH\\_NCR2\)](#)). 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.

### 4.6.4 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:

- In the main program memory:  
Block program operations to the main program memory have to be executed totally from RAM.
- In the DATA area
  - Devices with RWW capability: DATA block operations can be executed from the main program memory. However, the data loading phase (see below) has to be executed from RAM.
  - Devices without RWW capability: Block program operations must be 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 erased before being programmed.

To program a whole block in standard mode, the PRG/NPRG bits in the FLASH\_CR2 and FLASH\_NCR2 registers must be previously set/cleared to enable standard block programming (see [Section 4.8.2: Flash control register 2 \(FLASH\\_CR2\)](#) and [Section 4.8.2: Flash control register 2 \(FLASH\\_NCR2\)](#)). 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.

When programming a block in DATA area, the application can check the HVOFF bit in the [Flash status register \(FLASH\\_IAPSR\)](#). As soon the HVOFF flag is reset the actual programming phase starts and the application can return to main program memory.

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/NFPRG bits of the FLASH\_CR2 and FLASH\_NCR2 registers must be previously set/cleared.

The HVOFF flag can also be polled by the application which can execute other instructions (RWW) during the actual programming phase of the DATA.

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/NERASE bits in the FLASH\_CR2 and FLASH\_NCR2 registers must be previously set/cleared to enable block erasing (see [Section 4.8.2: Flash control register 2 \(FLASH\\_CR2\)](#) and [Section 4.8.3: Flash complementary control register 2 \(FLASH\\_NCR2\)](#)). 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
Low density STM8S and STM8AF	64 bytes
Medium density STM8S and STM8AF (up to 32 Kbytes)	128 bytes
High density STM8S and STM8AF (up to 128 Kbytes)	128 bytes

## 4.6.5 Option byte programming

Option byte programming is very similar to data EEPROM byte programming.

The application writes directly to the target address. The program does not stop and the write operation is performed using the RWW capability.

Refer to the datasheet for details of the option byte contents.

## 4.7 ICP (in-circuit programming) and IAP (in-application programming)

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, I2C, 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 STM8S and STM8AF Flash programming manual (PM0051) and STM8 SWIM protocol and debug manual (UM0470) for more information on programming procedures.

**Table 6. Memory access versus programming method<sup>(1)</sup>**

Mode	ROP	Memory Area	Access from core
User, IAP, and bootloader (if available)	Readout protection enabled	User boot code area (UBC)	R/E
		Main program	R/W/E <sup>(2)</sup>
		Data EEPROM area (DATA)	R/W <sup>(3)</sup>
		Option bytes	R
	Readout protection disabled	User boot code area (UBC)	R/E <sup>(4)</sup>
		Main program	R/W/E <sup>(2)</sup>
		Data EEPROM area (DATA)	R/W <sup>(3)</sup>
		Option bytes	R/W <sup>(5)</sup>
SWIM active (ICP mode)	Readout protection enabled	User boot code area (UBC)	P
		Main program	P
		Data EEPROM area (DATA)	P
		Option bytes	P/W <sub>ROP</sub> <sup>(6)</sup>
	Readout protection disabled	User boot code area (UBC)	R/E <sup>(4)</sup>
		Main program	R/W/E <sup>(2)</sup>
		Data EEPROM area (DATA)	R/W <sup>(3)</sup>
		Option bytes	R/W <sup>(5)</sup>

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<sub>ROP</sub> = Protected, write forbidden except for ROP option byte.
2. 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. If incorrect keys are provided, the device must be reset and new keys programmed.
3. 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 IAPSR register. If incorrect keys are provided, another key program sequence can be performed without resetting the device.
4. To program the UBC area, the application must first clear the UBC option byte.
5. 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.
6. When ROP is removed, the whole memory is erased, including the 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		HALT	AHALT	IE	FIX
		r		rw	rw	rw	rw

Bits 7:4 Reserved

Bit 3 **HALT**: Power-down in Halt mode

This bit is set and cleared by software.

0: Flash in power-down mode when MCU is in Halt mode

1: Flash in operating mode when MCU is in Halt mode

Bit 2 **AHALT**: Power-down in Active-halt mode

This bit is set and cleared by software.

0: Flash in operating mode when MCU is in Active-halt mode

1: Flash in power-down when MCU is in Active-halt mode

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		r		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<sup>(1)</sup>**: 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<sup>(1)</sup>**: 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 complementary control register 2 (FLASH\_NCR2)

Address offset: 0x02

Reset value: 0xFF

7	6	5	4	3	2	1	0
NOPT	NWPRG	NERASE	NFPRG	Reserved			NPRG
rw	rw	rw	rw		r		rw

Bit 7 **NOPT**: Write option bytes

This bit is set and cleared by software.

0: Write access to option bytes enabled

1: Write access to option bytes disabled

Bit 6 **NWPRG**: Word programming

This bit is cleared by software and set by hardware when the operation is completed.

0: Word programming enabled

1: Word programming disabled

Bit 5 **NERASE**: Block erase

This bit is cleared by software and set by hardware when the operation is completed.

0: Block erase enabled

1: Block erase disabled

Bit 4 **NFPRG**: Fast block programming

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

0: Fast block programming enabled (no erase before programming, the programmed data values are not guaranteed when the block is not blank (fully erased) before the operation)

1: Fast block programming disabled

Bits 3:1 Reserved.

Bit 0 **NPRG**: Block programming

This bit is cleared by software and set by hardware when the operation is completed.

0: Block programming enabled

1: Block programming disabled

#### 4.8.4 Flash protection register (FLASH\_FPR)

Address offset: 0x03

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	WPB5	WPB4	WPB3	WPB2	WPB1	WPB0	
r	ro	ro	ro	ro	ro	ro	ro

Bits 7:6 Reserved.

Bits 5:0 **WPB[5:0]**: User boot code area protection bits

These bits show the size of the boot code area. They are loaded at startup with the content of the UBC option byte. Refer to the datasheets for the protected pages according to the bit values.

#### 4.8.5 Flash protection register (FLASH\_NFPR)

Address offset: 0x04

Reset value: 0xFF

7	6	5	4	3	2	1	0
Reserved	NWPB5	NWPB4	NWPB3	NWPB2	NWPB1	NWPB0	
	ro	ro	ro	ro	ro	ro	ro

Bits 7:6 Reserved.

Bits 5:0 **WPB[5:0]**: User boot code area protection bits

These bits show the size of the boot code area. They reflect the content of the NUBC option byte. Refer to the datasheet for the protected pages according to the bit values.

#### 4.8.6 Flash program memory unprotected key register (FLASH\_PUKR)

Address offset: 0x08

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 44](#) for the description of main program area write unprotection mechanism.

#### 4.8.7 Data EEPROM unprotection key register (FLASH\_DUKR)

Address offset: 0x0A

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 45](#) for the description of main program area write unprotection mechanism.

#### 4.8.8 Flash status register (FLASH\_IAPSR)

Address offset: 0x05

Reset value: 0x40

7	6	5	4	3	2	1	0
Reserved	HVOFF	Reserved		DUL	EOP	PUL	WR_PG_DIS
res.	r	r		rc_w0	rc_r	rc_w0	rc_r

Bit 7 Reserved.

Bit 6 **HVOFF**: End of high voltage flag

This bit is set and cleared by hardware.

0: HV ON, start of actual programming

1: HV OFF, end of high voltage

Bits 5:4 Reserved, forced by hardware to 0.

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.

#### 4.8.9 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 and reset values**

Address	Register name	7	6	5	4	3	2	1	0
0x00	FLASH_CR1 Reset value	- 0	- 0	- 0	- 0	HALT 0	AHALT 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_NCR2 Reset value	NOPT 1	NWPRG 1	NERAS E 1	NFPRG 1	- 1	- 1	- 1	NPRG 1
0x03	FLASH_FPR Reset value	- 0	- 0	WPB5 0	WPB4 0	WPB3 0	WPB2 0	WPB1 0	WPB0 0
0x04	FLASH_NFPR Reset value	- 1	- 1	NWPB5 1	NWPB4 1	NWPB3 1	NWPB2 1	NWPB1 1	NWPB0 1
0x05	FLASH_IAPSR Reset value	- 0	HVOFF 1	- 0	- 0	DUL 0	EOP 0	PUL 0	WR_PG_DIS 0
0x06- 0x07		Reserved							
0x08	FLASH_PUKR Reset value	PUK7 0	PUK6 0	PUK5 0	PUK4 0	PUK3 0	PUK2 0	PUK1 0	PUK0 0
0x09		Reserved							
0x0A	FLASH_DUKR Reset value	DUK7 0	DUNP6 0	DUK5 0	DUK4 0	DUK3 0	DUK2 0	DUK1 0	DUK0 0

## 5 Single wire interface module (SWIM) and debug module (DM)

### 5.1 SWIM and DM introduction

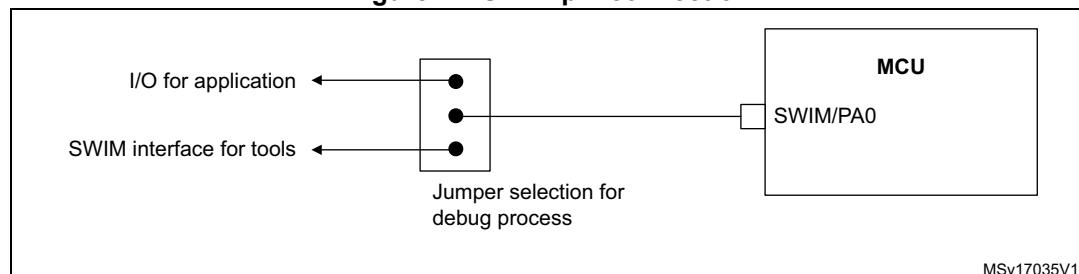
In-circuit debugging mode or in-circuit programming mode are managed through a single wire hardware interface featuring ultrafast memory programming. Coupled with an in-circuit debugging module, it also offers a non-intrusive emulation mode, making the in-circuit debugger extremely powerful, close in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

### 5.2 SWIM 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)*.
- Clock speed selectable in the *SWIM clock control register (CLK\_SWIMCCR)*.

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 12. SWIM pin connection**



MSv17035V1

### 5.3 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.

There are two important considerations to highlight for the devices where the NRST pin is not present:

- If the SWIM pin should be used with the I/O pin functionality, it is recommended to add a ~5 seconds delay in the firmware before changing the functionality on the pin with SWIM functions. This action allows the user to set the device into SWIM mode after the device power on and to be able to reprogram the device. If the pin with SWIM functionality is set to I/O mode immediately after the device reset, the device is unable to connect through the SWIM interface and it will be locked forever (if the NRST pin is not available on the package). This initial delay can be removed in the final (locked) code.
- Their program memory must contain a valid program loop. If the device's memory is empty, the program continues into non-existing memory space and executes invalid opcode; this causes the device to reset (reading of non-existing memory is random content). This behavior might lead to periodic device resets and to a difficulty to connect to the device through the SWIM interface.

## 6 Interrupt controller (ITC)

### 6.1 ITC introduction

- Management of hardware interrupts
  - External interrupt capability on most I/O pins with dedicated interrupt vector and edge sensitivity setting per port
  - 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
  - Up to 32 interrupt vectors fixed by hardware
  - 2 non maskable events: RESET, TRAP
  - 1 non-maskable top level hardware interrupt (TLI)

This interrupt management is based on:

- Bit I1 and I0 of the CPU Condition Code register (CCR)
- Software priority registers (ITC\_SPRx)
- Reset vector address 0x00 8000 at the beginning of program memory. In devices with boot ROM, 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.

### 6.2 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 13](#).

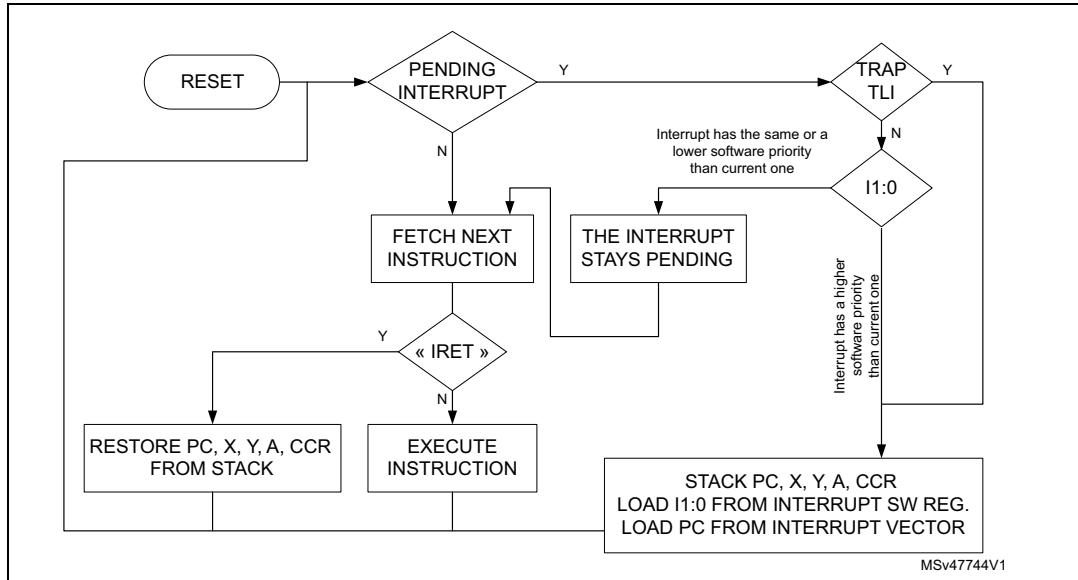
When an interrupt request has to be serviced:

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 .

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 13. 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
<pre>#asm PUSH CC POP ISR_CC<sup>(1)</sup> SIM #endasm</pre>	<pre>#asm PUSH ISR_CC<sup>(1)</sup> POP CC #endasm</pre>

1. ISR\_CC is a variable which stores the current value of the CC register.

## 6.2.1 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:

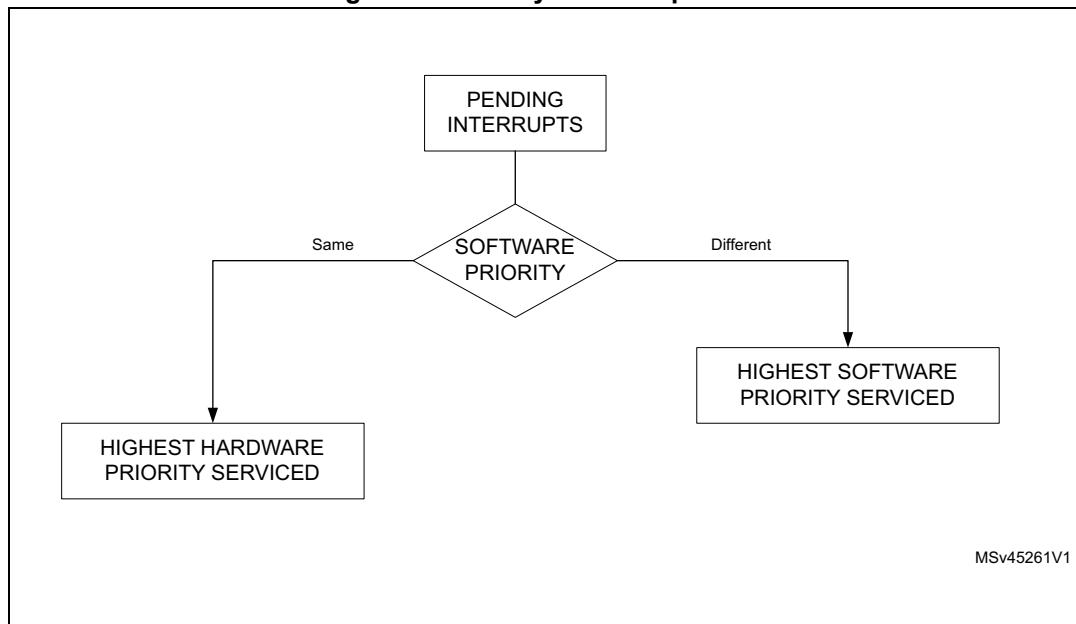
1. The highest software priority interrupt is serviced.
2. If several interrupts have the same software priority then the interrupt with the highest hardware priority is serviced first.

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, TLI and TRAP are considered as having the highest software priority in the decision process.*

See [Figure 14](#) for a description of pending interrupt servicing process.

**Figure 14. Priority decision process**



## 6.2.2 Interrupt sources

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

- Non-maskable interrupts: RESET, TLI 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 13](#)). PC, X, Y, A and CCR registers are stacked only when a TRAP

or TLI 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 accordingly to the flowchart shown in [Figure 13](#).

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 11: 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.

- TLI (top level hardware interrupt)

This hardware interrupt occurs when a specific edge is detected on the corresponding TLI input. It is serviced accordingly to the flowchart shown in [Figure 13](#).

**Caution:** A TRAP instruction must not be used in a TLI service routine.

### 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

Most 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.

## 6.3 Interrupts and low power modes

All interrupts allow the processor to exit from Wait mode.

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 in the interrupt vector table in the datasheet).

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 14](#). 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.

## 6.4 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 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 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.

## 6.5 Concurrent and nested interrupt management

STM8 devices feature two interrupt management modes:

- Concurrent mode
- Nested mode

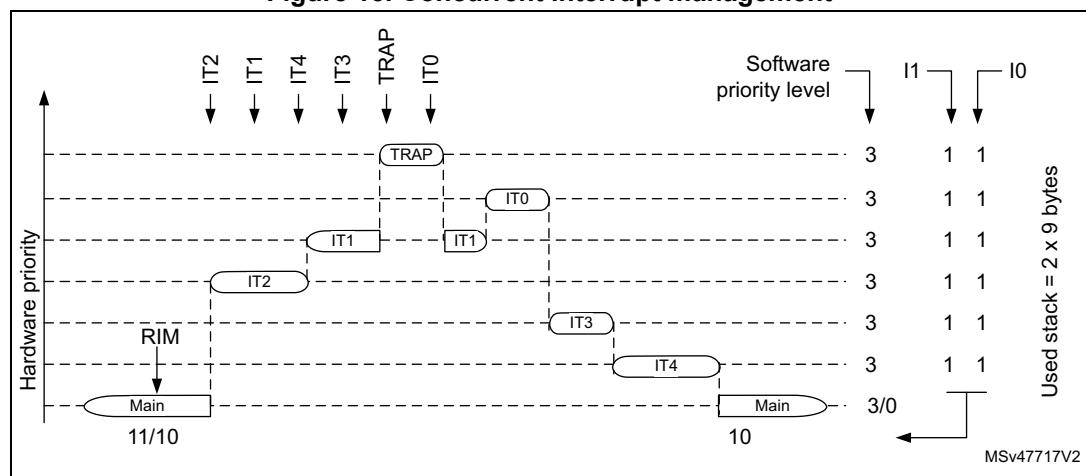
### 6.5.1 Concurrent interrupt management mode

In this mode, all interrupts are interrupt priority level 3 so that none of them can be interrupted, except by a TLI, 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/TLI (same priority), and RESET.

*Figure 15* shows an example of concurrent interrupt management mode.

**Figure 15. Concurrent interrupt management**



## 6.5.2 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 6.7](#) for the list of dedicated interrupt instructions.

[Figure 16](#) shows an example of nested interrupt management mode.

---

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

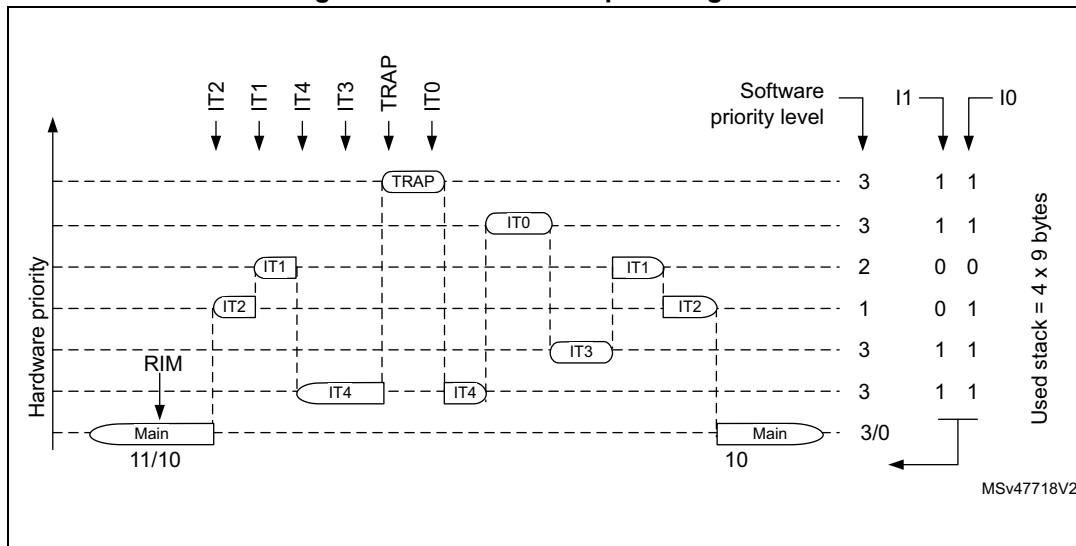
---

**Table 10. Vector address map versus software priority bits**

Vector address	ITC_SPRx bits
0x00 8008h	I1_0 and I0_0 bits <sup>(1)</sup>
0x00 800Ch	I1_1 and I0_1 bits
...	...
0x00 807Ch	I1_29 and I0_29 bits

1. ITC\_SPRx register bits corresponding to the TLI can be read and written. However they are not significant in the interrupt process management.

Figure 16. Nested interrupt management



## 6.6 External interrupts

Five interrupt vectors are dedicated to external Interrupt events:

- 5 lines on Port A: PA[6:2]
- 8 lines on Port B: PB[7:0]
- 8 lines on Port C: PC[7:0]
- 7 lines on Port D: PD[6:0]
- 8 lines on Port E: PE[7:0]

PD7 is the Top Level Interrupt source (TLI), except for 20-pin packages on which the Top Level Interrupt source (TLI) can be available on the PC3 pin using an alternate function remapping option bit. Refer to option bytes section in the product datasheet for more details.

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.

The interrupt sensitivity must be configured in the external interrupt control register 1 (EXTI\_CR1) and external interrupt control register 2 (EXTI\_CR2) (see [Section 6.9.3](#) and [Section 6.9.4](#)).

## 6.7 Interrupt instructions

[Table 11](#) shows the interrupt instructions.

Table 11. 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 ?	-	-	-	-	-	-

**Table 11. Dedicated interrupt instruction set (continued)**

Instruction	New description	Function/example	I1	H	I0	N	Z	C
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	-	-	-	-	-	-
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	-	-	-

## 6.8 Interrupt mapping

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

## 6.9 ITC and EXTI registers

### 6.9.1 CPU condition code register interrupt bits (CCR)

Address: refer to the general hardware register map table in the datasheet.

Reset value: 0x28

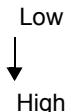
7	6	5	4	3	2	1	0
V	-	I1	H	I0	N	Z	C
r	r	rw	r	rw	r	r	r

Bits 5, 3<sup>(1)</sup> **I[1:0]**: Software interrupt priority bits<sup>(2)</sup>

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 16: Nested interrupt management](#)).

I1	I0	Priority	Level
1	0	Level 0 (main)	
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. TLI, TRAP and RESET events can interrupt a level-3 program.

## 6.9.2 Software priority register x (ITC\_SPRx)

Address offset: 0x00 to 0x07

Reset value: 0xFF

	7	6	5	4	3	2	1	0
ITC_SPR1	VECT3SPR[1:0]		VECT2SPR[1:0]		VECT1SPR[1:0]		VECT0SPR[1:0]	
ITC_SPR2	VECT7SPR[1:0]		VECT6SPR[1:0]		VECT5SPR[1:0]		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]		VECT18SPR[1:0]		VECT17SPR[1:0]		VECT16SPR[1:0]	
ITC_SPR6	VECT23SPR[1:0]		VECT22SPR[1:0]		VECT21SPR[1:0]		VECT20SPR[1:0]	
ITC_SPR7	VECT27SPR[1:0]		VECT26SPR[1:0]		VECT25SPR[1:0]		VECT24SPR[1:0]	
ITC_SPR8	Reserved				VECT29SPR[1:0]		VECT28SPR[1:0]	
	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 6.9.1: CPU condition code register interrupt bits \(CCR\)](#) for the values to be programmed for each priority.

ITC\_SPR1 bits 1:0 are forced to 1 by hardware (TLI)

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.*

### 6.9.3 External interrupt control register 1 (EXTI\_CR1)

Address offset: 0x00

Reset value: 0x00

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

#### Bits 7:6 **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 Port 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 **PCIS[1:0]:** Port C 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 Port C external interrupts.

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

#### Bits 3:2 **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 Port B external interrupts.

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

#### Bits 1:0 **PAIS[1:0]:** Port A 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 Port A external interrupts.

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

### 6.9.4 External interrupt control register 1 (EXTI\_CR2)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				TLIS		PEIS[1:0]	
r				rw		rw	

Bits 7:3 Reserved.

Bit 2 **TLIS**: Top level interrupt sensitivity

This bit is set and cleared by software. This bit can be written only when external interrupt is disabled on the corresponding GPIO port (PD7 or PC3, refer to [Section 6.6: External interrupts on page 66](#)).

0: Falling edge

1: Rising edge

Bits 1:0 **PEIS[1:0]**: Port E 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 E external interrupts.

00: Falling edge and low level

01: Rising edge only

10: Falling edge only

11: Rising and falling edge

## 6.9.5 ITC and EXTI register map and reset values

**Table 12. Interrupt register map**

Add. offset	Register name	7	6	5	4	3	2	1	0
<b>ITC-SPR block<sup>(1)</sup></b>									
0x00	ITC_SPR1	VECT3 SPR1	VECT3 SPR0	VECT2 SPR1	VECT2 SPR0	VECT1 SPR1	VECT1 SPR0	Reserved	Reserved
	Reset value	1	1	1	1	1	1	1	1
0x01	ITC_SPR2	VECT7 SPR1	VECT7 SPR0	VECT6 SPR1	VECT6 SPR0	VECT5 SPR1	VECT5 SPR0	VECT4 SPR1	VECT4 SPR0
	Reset value	1	1	1	1	1	1	1	1
0x02	ITC_SPR3	VECT11 SPR1	VECT11 SPR0	VECT10 SPR1	VECT10 SPR0	VECT9 SPR1	VECT9 SPR0	VECT8 SPR1	VECT8 SPR0
	Reset value	1	1	1	1	1	1	1	1
0x03	ITC_SPR4	VECT15 SPR1	VECT15 SPR0	VECT14 SPR1	VECT14 SPR0	VECT13 SPR1	VECT13 SPR0	VECT12 SPR1	VECT12 SPR0
	Reset value	1	1	1	1	1	1	1	1
0x04	ITC_SPR5	VECT19 SPR1	VECT19 SPR0	VECT18 SPR1	VECT18 SPR0	VECT17 SPR1	VECT17 SPR0	VECT16 SPR1	VECT16 SPR0
	Reset value	1	1	1	1	1	1	1	1
0x05	ITC_SPR6	VECT23 SPR1	VECT23 SPR0	VECT22 SPR1	VECT22 SPR0	VECT21 SPR1	VECT21 SPR0	VECT20 SPR1	VECT20 SPR0
	Reset value	1	1	1	1	1	1	1	1
0x06	ITC_SPR7	VECT27 SPR1	VECT27 SPR0	VECT26 SPR1	VECT26 SPR0	VECT25 SPR1	VECT25 SPR0	VECT24 SPR1	VECT24 SPR0
	Reset value	1	1	1	1	1	1	1	1
0x07	ITC_SPR8	-	-	-	-	-	-	VECT28 SPR1	VECT28 SPR0
	Reset value	-	-	-	-	-	-	1	1
<b>ITC-EXTI block<sup>(2)</sup></b>									
0x00	EXTI_CR1	PDIS1	PDIS0	PCIS1	PCIS0	PBIS1	PBIS0	PAIS1	PAIS0
	Reset value	0	0	0	0	0	0	0	0
0x01	EXTI_CR2	-	-	-	-	-	TLIS	PEIS1	PEIS0
	Reset value	0	0	0	0	0	0	0	0

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

## 7 Power supply

The MCU has four distinct power supplies:

- $V_{DD}/V_{SS}$ : Main power supply (3 V to 5.5 V)
- $V_{DDIO}/V_{SSIO}$ : I/O power supply (3 V to 5.5 V)
- $V_{DDA}/V_{SSA}$ : Power supply for the analog functions
- $V_{REF+}/V_{REF-}$ : Reference supply for Analog Digital Converter

The  $V_{DD}/V_{SS}$  pins are used to supply the internal Main Voltage Regulator (MVR) and the internal Low Power Voltage Regulator (LPVR). The 2 regulator outputs are connected and provide the 1.8 V supply ( $V_{18}$ ) to the MCU core (CPU, Flash and RAM)

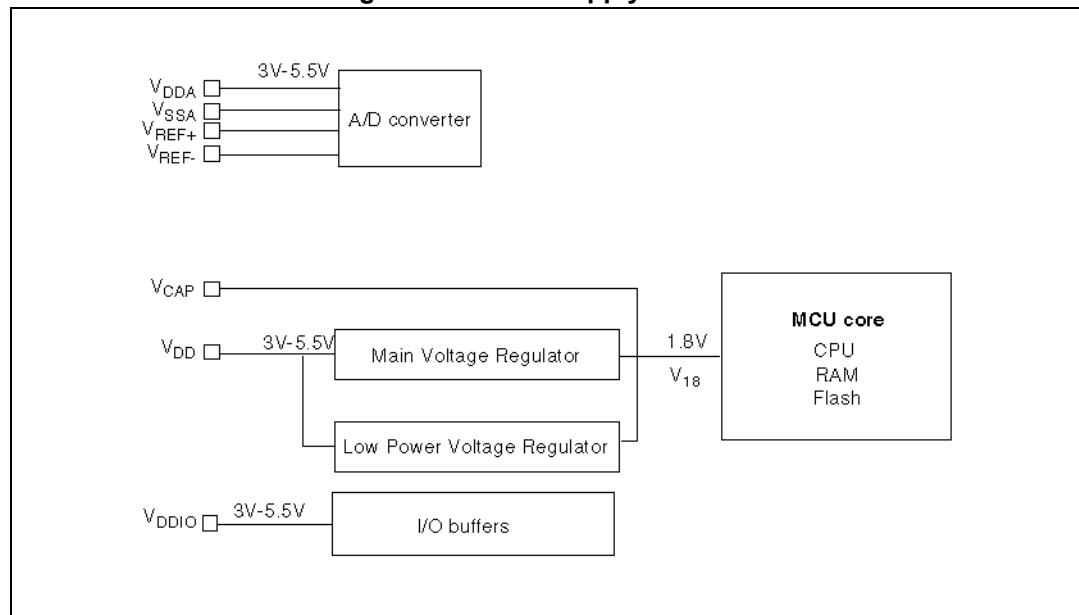
In low power modes the system automatically switches from the MVR to the LPVR in order to reduce current consumption.

To stabilize the MVR, a capacitor must be connected to the VCAP pin (for more details refer to the datasheet electrical characteristics section).

Depending on the package size, there are one or two pairs of dedicated pins for  $V_{DDIO}/V_{SSIO}$  to supply power to the I/Os.

$V_{DDA}/V_{SSA}$  and  $V_{REF+}/V_{REF-}$  are connected to the Analog to Digital Converter (ADC).

Figure 17. Power supply overview



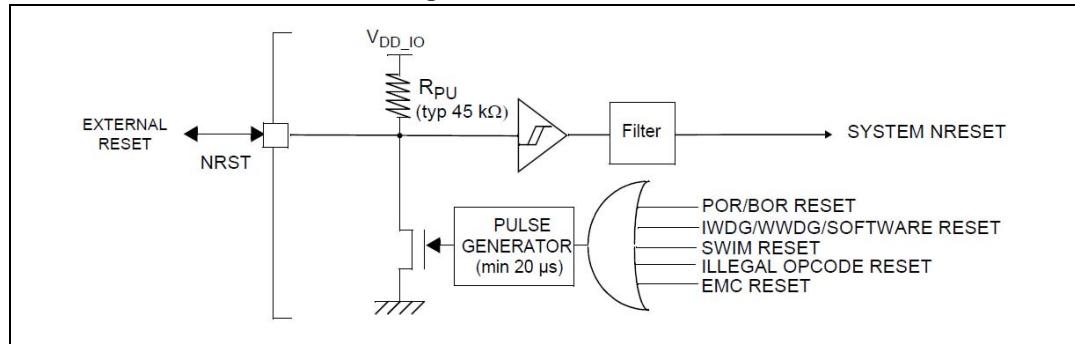
## 8 Reset (RST)

There are 9 reset sources:

- External reset through the NRST pin
- Power-on reset (POR)
- Brown-out Reset (BOR)
- Independent watchdog reset (IWDG)
- Window watchdog reset (WWDG)
- Software reset
- SWIM reset
- Illegal opcode reset
- EMC reset: generated if critical registers are corrupted or badly loaded

These sources act on the RESET pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at address 6000h in the memory map.

**Figure 18. Reset circuit**



### 8.1 “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.

### 8.2 Reset circuit description

The NRST pin is both an input and an open-drain output with integrated R<sub>PU</sub> weak pull-up resistor.

The low pulse of duration t<sub>INFP(NRST)</sub> on the NRST pin generates 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 duration  $t_{OP(NRST)}$  whatever the internal reset source. An additional internal weak pull-up ensures a high level on the reset pin when the reset is not forced.

## 8.3 Internal reset sources

Each internal reset source is linked to a specific flag bit in the [Reset status register \(RST\\_SR\)](#) except POR/BOR which have no flag. These flags are set respectively at reset 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”.

### 8.3.1 Power-on reset (POR) and brown-out reset (BOR)

During power-on, the POR keeps the device under reset until the supply voltages ( $V_{DD}$  and  $V_{DDIO}$ ) reach the voltage level at which the BOR starts to function. At this point, the BOR reset replaces the POR and the POR is automatically switched off. The BOR reset is maintained till the supply voltage reaches the operating voltage range.

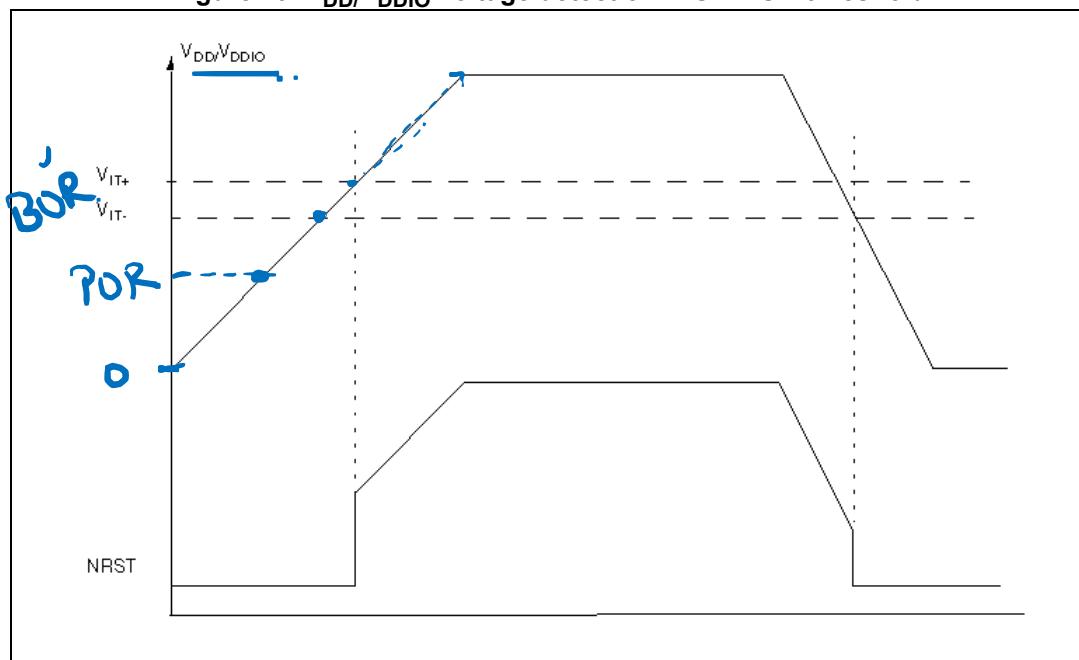
See Electrical parameters section of the datasheet for more details.

The BOR also generates a reset when the supply voltage drops below the  $V_{IT-}$  threshold. When this occurs, the POR is re-armed for the next power-on phase.

An hysteresis is implemented to ensure clean detection of voltage rise and fall.

The BOR always remains active even when the MCU is put into Low Power mode.

**Figure 19.  $V_{DD}/V_{DDIO}$  voltage detection: POR/BOR threshold**



### 8.3.2 Watchdog reset

Refer to [Section 15: Window watchdog \(WWDG\)](#) and [Section 14: Independent watchdog \(IWDG\)](#) for details.

### 8.3.3 Software reset

The application software can trigger reset by clearing bit T6 in the WWDG\_CR register. Refer to [Section 15: Window watchdog \(WWDG\)](#).

### 8.3.4 SWIM reset

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

### 8.3.5 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.

*Note:* A valid prebyte associated with a valid opcode forming an unauthorized combination does not generate a reset.

### 8.3.6 EMC reset

To protect the application against spurious write access or system hang-up, possibly caused by electromagnetic disturbance, the most critical registers are implemented as two bitfields that must contain complementary values. Mismatches are automatically detected by this mechanism, triggering an EMC reset and allowing the application to cleanly recover normal operations.

## 8.4 RST register description

### 8.4.1 Reset status register (RST\_SR)

Address offset: 0x00

Reset value: 0XX

7	6	5	4	3	2	1	0
Reserved		EMCF	SWIMF	ILLOPF	IWDGF	WWDDGF	
r		rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 7:5 Reserved.

Bit 4 **EMCF**: EMC reset flag

This bit is set by hardware and cleared by software writing “1”.

0: No EMC reset occurred

1: An EMC reset occurred (possible cause: complementary register or option byte mismatch).

Bit 3 **SWIMF**: SWIM reset flag

This bit is set by hardware and cleared by software writing “1”.

0: No SWIM reset occurred

1: A SWIM reset occurred

Bit 2 **ILLOPF**: Illegal opcode reset flag

This bit is set by hardware and cleared by software writing “1”.

0: No ILLOP reset occurred

1: An ILLOP reset occurred

Bit 1 **IWDGF**: Independent Watchdog reset flag

This bit is set by hardware and cleared by software writing “1”.

0: No IWDG reset occurred

1: An IWDG reset occurred

Bit 0 **WWDDGF**: Window Watchdog reset flag

This bit is set by hardware and cleared by software writing “1”.

0: No WWDDG reset occurred

1: An WWDDG reset occurred

## 8.5 RST register map

Refer to the corresponding datasheet for the base address.

Table 13. RST register map

Address offset	Register Name	7	6	5	4	3	2	1	0
0x00	RST_SR Reset value	- x	- x	- x	EMCF x	SWIMF x	ILLOPF x	IWDGF x	WWDDGF x

## 9 Clock control (CLK)

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

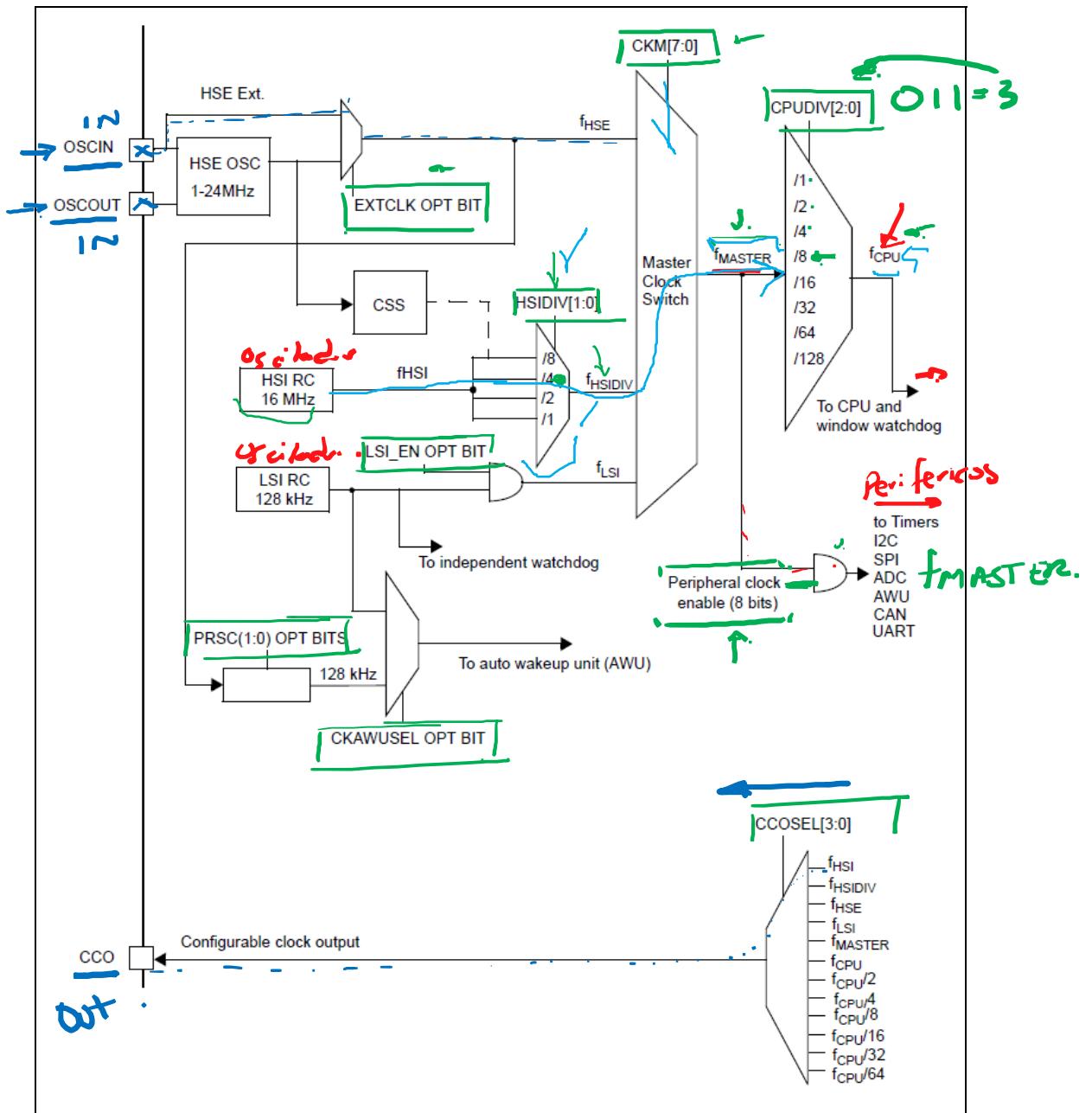
You can manage all the different clock sources independently and distribute them to the CPU and to the various peripherals. Prescalers are available for the master and CPU clocks.

A safe and glitch-free switch mechanism allows you to switch the master clock on the fly from one clock source to another one.

### EMC-hardened clock configuration registers

To protect the application against spurious write access or system hang-up, possibly caused by electromagnetic disturbance, the most critical CLK registers are implemented as two bitfields that must contain complementary values. Mismatches are automatically detected by the CLK, triggering an EMC reset and allowing the application to cleanly recover normal operations. See [CLK register description](#) for more details.

Figure 20. Clock tree



1. Legend: HSE = High speed external clock signal; HSI = High speed internal clock signal; LSI = Low Speed internal clock signal.

## 9.1 Master clock sources

4 different clock sources can be used to drive the master clock:

- 1-24 MHz high speed external crystal oscillator (HSE)
- Up to 24 MHz high speed user-external clock (HSE user-ext)
- → 16 MHz high speed internal RC oscillator (HSI)
- 128 kHz low speed internal RC (LSI)

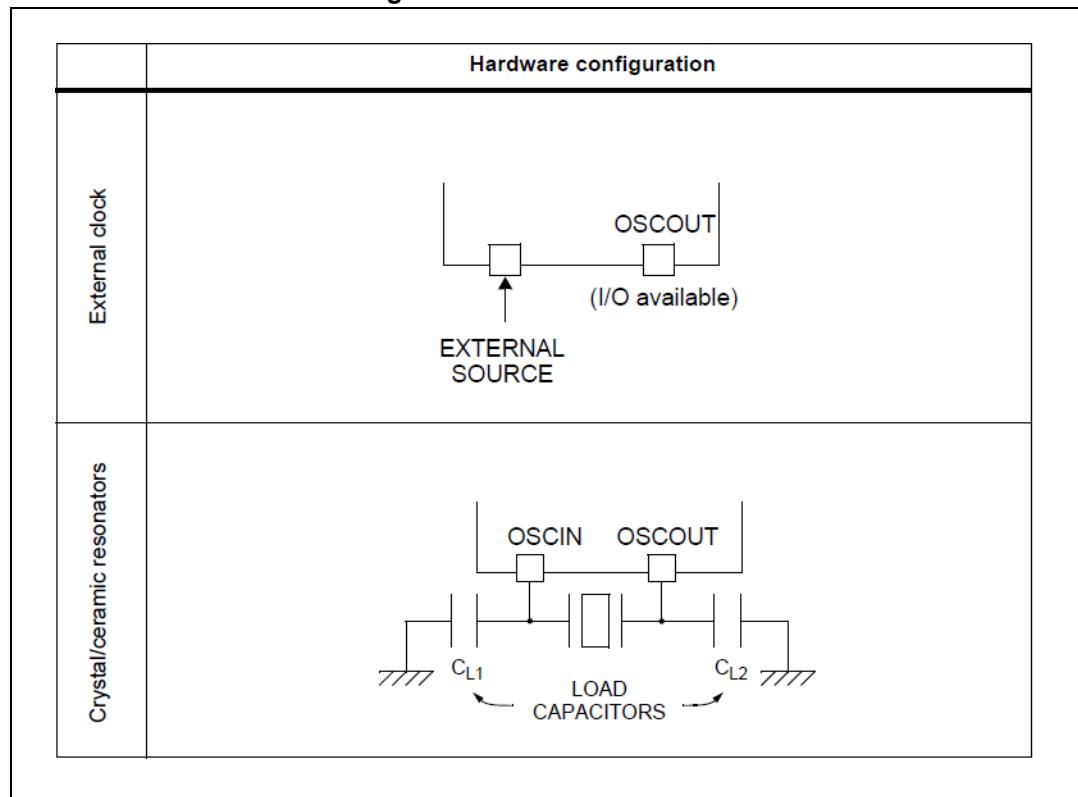
Each clock source can be switched on or off independently when it is not used, to optimize power consumption.

### 9.1.1 HSE (high-speed external) clock signal

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

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

**Figure 21. HSE clock sources**



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

### External crystal/ceramic resonator (HSE crystal)

The 1 to 24 MHz external oscillator has the advantage of producing a very accurate rate on the main clock with 50% duty cycle.

The associated hardware configuration is shown in [Figure 21](#). Refer to the electrical characteristics section for more details.

At start up the clock signal produced by the oscillator is not stable, and by default a delay of 2048 osc cycles is inserted before the clock signal is released. You can program a shorter stabilization time in the HSECNT option byte, please refer to option bytes section in the datasheet.

The HSERDY flag in the [External clock register \(CLK\\_ECKR\)](#) indicates if the high-speed external oscillator is stable or not. At startup, the clock is not released until this bit is set by hardware.

The HSE Crystal can be switched on and off using the HSEEN bit in the [External clock register \(CLK\\_ECKR\)](#).

#### External source (HSE user-ext)

In this mode, an external clock source must be provided. It can have a frequency of up to 24 MHz. You select this mode by programming the EXTCLK option bit. Refer to the option bytes section of the datasheet. The external clock signal (square, sinus or triangle) with ~50% duty cycle has to drive the OSCIN pin while the OSCOUT pin is available as standard I/O (see [Figure 20](#)).

**Note:** For clock frequencies above 16 MHz, Flash /data EEPROM access must be configured for 1 wait state. This is enabled by the device option byte. Refer to the datasheet option byte section.

### 9.1.2 HSI (high-speed internal) clock signal



The HSI clock signal is generated from an internal 16 MHz RC oscillator together with a programmable divider (factor 1 to 8). 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 ( $f_{HSI}/8$ ).

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). It also has a faster startup time than the HSE crystal oscillator however, even with calibration the frequency is less accurate than an external crystal oscillator or ceramic resonator.

The HSIRDY flag in the [Internal clock register \(CLK\\_ICKR\)](#) indicates if the HSI RC is stable or not. At startup, the HSI RC output clock is not released until this bit is set by hardware.

The HSI RC can be switched on and off using the HSIEN bit in the [Internal clock register \(CLK\\_ICKR\)](#).

#### Backup source

The HSI/8 signal can also be used as a backup source (Auxiliary clock) if the HSE crystal oscillator fails. Refer to [Section 9.6: Clock security system \(CSS\)](#).

### Fast wakeup feature

If the FHWU bit in the [Internal clock register \(CLK\\_ICCR\)](#) is set, this automatically selects the HSI clock as master clock after MCU wakeup from Halt or Active-halt (see Low power chapter).

### Calibration

Each device is factory calibrated by ST.

After reset, the factory calibration value is automatically loaded in an internal calibration register.

If the application is subject to voltage or temperature variations this may affect the RC oscillator speed. You can trim the HSI frequency in the application using the [HSI clock calibration trimming register \(CLK\\_HSITRIMR\)](#). In this register there are 3 or 4 bits providing an additional trimming value that is added to the internal HSI calibration register value.

The width of the trimming steps with 4 bits is half the trimming step width with 3 bits.

**Table 14. Devices with 4 trimming bits**

Trimming bits value	Trimming steps	Trimming bits value	Trimming steps
0111b	+7	1111b	-1
0110b	+6	1110b	-2
0101b	+5	1101b	-3
0100b	+4	1100b	-4
0011b	+3	1011b	-5
0010b	+2	1010b	-6
0001b	+1	1001b	-7
0000b	0	1000b	-8

**Table 15. Devices with 3 trimming bits**

Trimming bits value	Trimming steps	Trimming bits value	Trimming steps
011b	+3	111b	-1
010b	+2	110b	-2
001b	+1	101b	-3
000b	0	100b	-4

As the trimming step width depends on the absolute frequency of the RC oscillator, a successive approximation method needs to be applied for the trimming. This method is described in a separate technical document.

### 9.1.3 LSI

The 128 kHz LSI RC acts as a low power, low cost alternative master clock source as well as a low power clock source that can be kept running in Halt mode for the independent watchdog (IWDG) and auto-wakeup unit (AWU).

The LSI RC can be switched on and off using the LSIEN bit in the *Internal clock register (CLK\_ICKR)*.

The LSIRDY flag in the *Internal clock register (CLK\_ICKR)* indicates if the low-speed internal oscillator is stable or not. At startup, the clock is not released until this bit is set by hardware.

### Calibration

Like the HSI RC, the LSI RC device is factory calibrated by ST. However, it is not possible to perform further trimming.

**Note:** When using the independent watchdog with the LSI as clock source, in order to guarantee that the CPU will never run on the same clock in case of corruption, the LSI clock cannot be the master clock if LSI\_EN option bit is reset. Refer to the option bytes section in the datasheet.

## 9.2 Master clock switching

The clock switching feature provides an easy to use, fast and secure way for the application to switch from one master clock source to another.

### 9.2.1 System startup

→ For fast system startup, after a reset the clock controller configures the master clock source as HSI RC clock output divided by 8 (HSI/8). This is to take advantage of the short stabilization time of the HSI oscillator. The /8 divider is to ensure safe start-up in case of poor V<sub>DD</sub> conditions.

Once the master clock is released, the user program can switch the master clock to another clock source.

### 9.2.2 Master clock switching procedures

To switch clock sources, you can proceed in one of two ways:

- Automatic switching
- Manual switching

#### Automatic switching

→ The automatic switching enables, the user to launch a clock switch with a minimum number of instructions. The software can continue doing other operations without taking care of the switch event exact time.

To enable automatic switching, follow the sequence below (refer to the flowchart in *Figure 22*):

1. Enable the switching mechanism by setting the SWEN bit in the *Switch control register (CLK\_SWCR)*.
2. Write the 8-bit value used to select the target clock source in the *Clock master switch register (CLK\_SWR)*. The SWBSY bit in the CLK\_SWCR register is set by hardware, and the target source oscillator starts. The old clock source continues to drive the CPU and peripherals.

As soon as the target clock source is ready (stabilized), the content of the CLK\_SWR register is copied to the [Clock master status register \(CLK\\_CMSR\)](#).

The SWBSY bit is cleared and the new clock source replaces the old one. The SWIF flag in the CLK\_SWCR is set and an interrupt is generated if the SWIEN bit is set.

### Manual switching

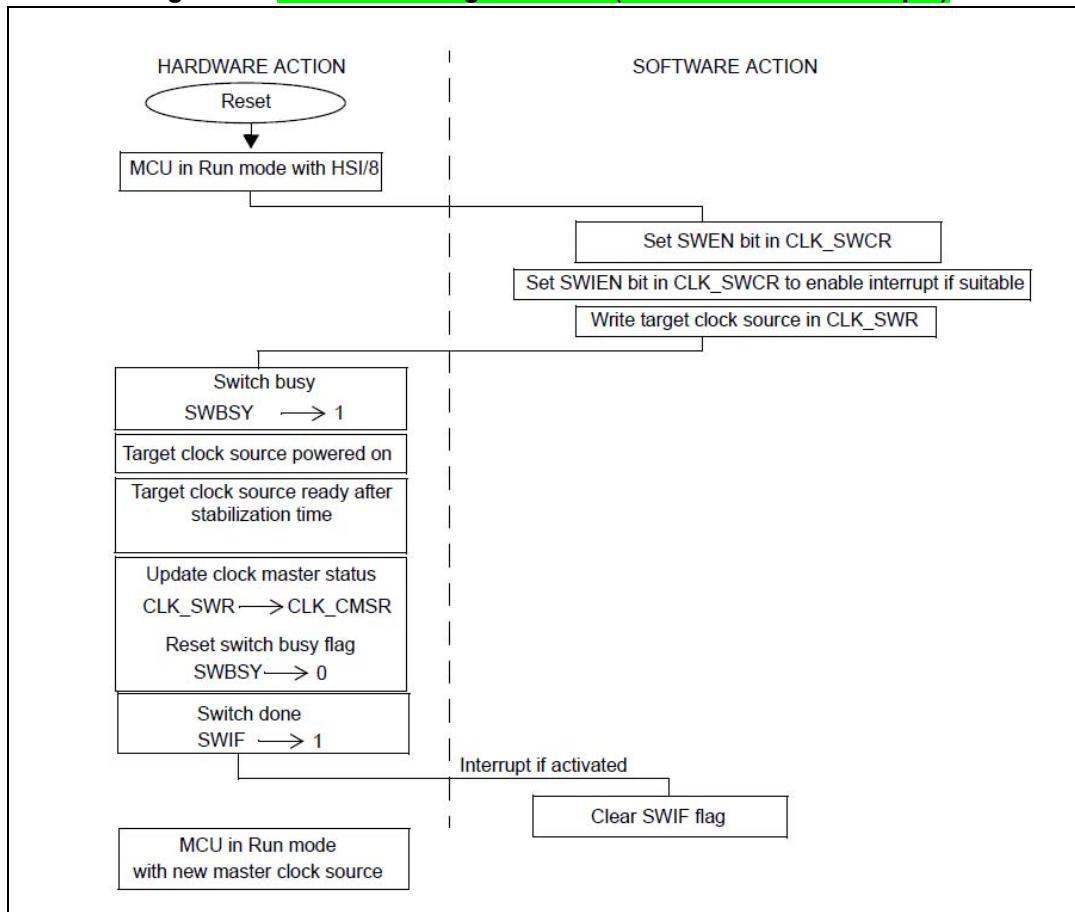
The manual switching is not as immediate as the automatic switching but it offers to the user a precise control of the switch event time.

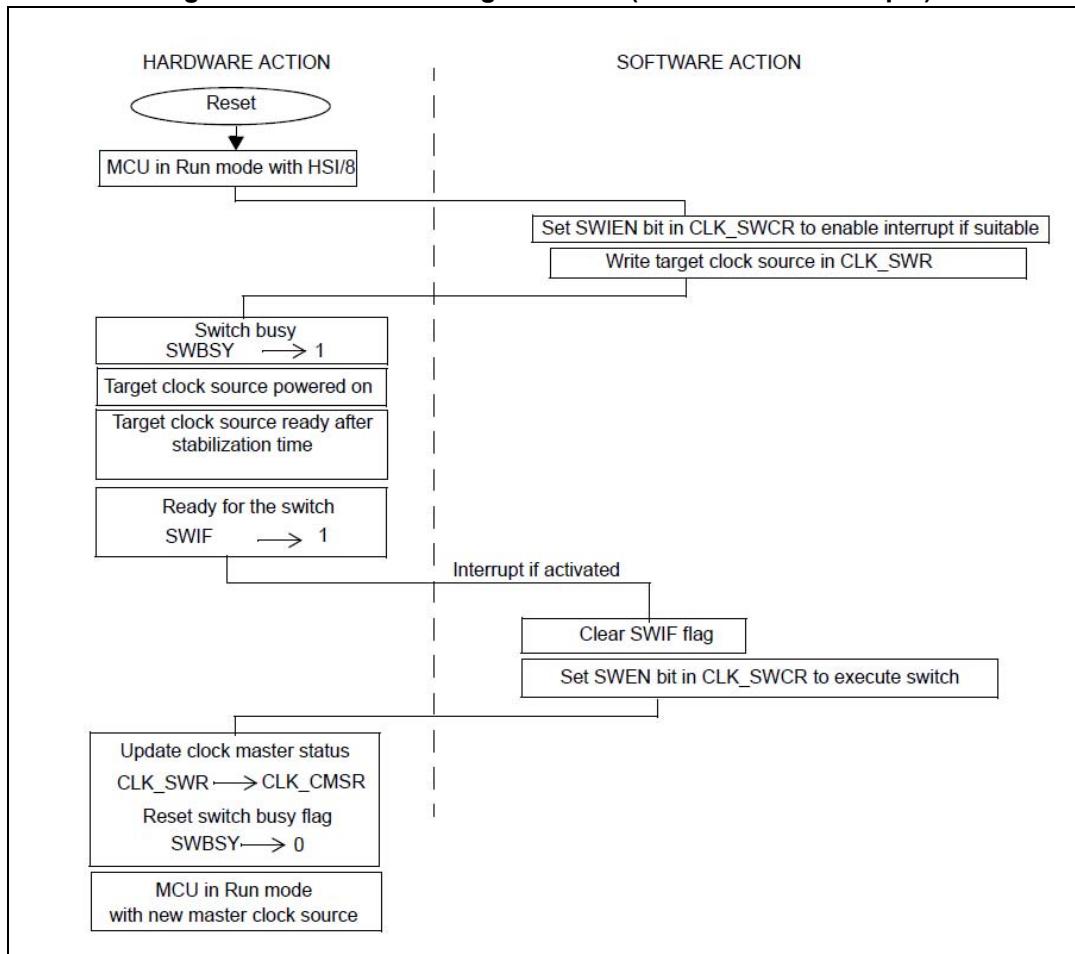
To enable manual switching, follow the sequence below (refer to the flowchart in [Figure 23](#)):

1. Write the 8-bit value used to select the target clock source in the [Clock master switch register \(CLK\\_SWR\)](#). Then the SWBSY bit is set by hardware, and the target source oscillator starts. The old clock source continues to drive the CPU and peripherals.
2. The software has to wait until the target clock source is ready (stabilized). This is indicated by the SWIF flag in the CLK\_SWCR register and by an interrupt if the SWIEN bit is set.
3. The final software action is to set, at the chosen time, the SWEN bit in the CLK\_SWCR register to execute the switch.

In both manual and automatic switching modes, the old master clock source will not be powered off automatically in case it is required by other blocks (the LSI RC may be used to drive the independent watchdog for example). The clock source can be powered off using the bits in the [Internal clock register \(CLK\\_ICKR\)](#) and [External clock register \(CLK\\_ECKR\)](#).

If the clock switch does not work for any reason, software can reset the current switch operation by clearing the SWBSY flag. This will restore the CLK\_SWR register to its previous content (old master clock).

**Figure 22. Clock switching flowchart (automatic mode example)**

**Figure 23. Clock switching flowchart (manual mode example)**

### 9.3 Low-speed clock selection

The Low-speed clock source for the AWU or the independent watchdog can be LSI or HSE divided according to the CKAWUSEL option bit. Refer to option bytes section in the datasheet.

The division factor for HSE has to be programmed in the HSEPRSC[1:0] option bits Refer to in the option bytes section of the datasheet. The goal is to get 128 kHz at the output of the HSE prescaler.

### 9.4 CPU clock-divider

The CPU clock ( $f_{CPU}$ ) is derived from the master clock ( $f_{MASTER}$ ), divided by a factor programmed in the CPUDIV[2:0] bits in the [Clock divider register \(CLK\\_CKDIVR\)](#). Seven division factors (1 to 128 in steps of power of 2) can be selected (refer to [Figure 20](#)).

The  $f_{CPU}$  signal is the clock for both the CPU and the window watchdog.

## 9.5 Peripheral clock-gating (PCG)

Gating the clock to unused peripherals helps reduce power consumption. Peripheral clock Gating (PCG) mode allows you to selectively enable or disable the  $f_{MASTER}$  clock connection to the following peripherals at any time in Run mode:

- ADC
- I2C
- AWU (register clock, not counter clock)
- SPI
- TIM[4:1]
- UART
- CAN (register clock, not CAN clock)

After a device reset, all peripheral clocks are enabled. You can disable the clock to any peripheral by clearing the corresponding PCKEN bit in the *Peripheral clock gating register 1 (CLK\_PCKENR1)* and in the *Peripheral clock gating register 2 (CLK\_PCKENR2)*. But you have to disable properly the peripheral using the appropriate bit, before stopping the corresponding clock.

To enable a peripheral, you must first enable the corresponding PCKEN bit in the CLK\_PCKENR registers and then set the peripheral enable bit in the peripheral's control registers.

The AWU counter is driven by an internal or external clock (LSI or HSE) independent from  $f_{MASTER}$ , so that it continues to run even if the register clock to this peripheral is switched off.

## 9.6 Clock security system (CSS)

The Clock Security System (CSS) monitors HSE crystal clock source failures. When f<sub>MASTER</sub> depends on HSE crystal, i.e. when HSE is selected, if the HSE clock fails due to a broken or disconnected resonator or any other reason, the clock controller activates a stall-safe recovery mechanism by automatically switching f<sub>MASTER</sub> to the auxiliary clock source (HSI/8). Once selected the auxiliary clock source remains enabled until the MCU is reset.

You enable the clock security system by setting the CSSEN bit in the [Clock security system register \(CLK\\_CSSR\)](#). For safety reason, once CSS is enabled it cannot be disabled until the next reset.

The following conditions must be met so that the CSS can detect HSE quartz crystal failures:

- HSE crystal on: (HSEEN = 1 in the [External clock register \(CLK\\_ECKR\)](#))
- HSE oscillator in quartz crystal configuration (EXTCLK option bit is reset)
- CSS function enabled: (CSSEN = 1 in the CLK\_CSSR register)

If HSE is the current clock master when a failure is detected, the CSS performs the following actions:

- The CSSD bit is set in the CLK\_CSSR register and an interrupt is generated if the CSSIEN bit is set.
- The [Clock master status register \(CLK\\_CMSR\)](#), [Clock master switch register \(CLK\\_SWR\)](#) register and the HSIDIV[1:0] bits in the [Clock divider register \(CLK\\_CKDIVR\)](#) are set to their reset values (CKM[7:0]= SWI[7:0]=E1h). HSI/8 becomes the master clock.
- The HSIEN bit in the [Internal clock register \(CLK\\_ICKR\)](#) register is set (HSI on).
- The HSEEN bit in the [External clock register \(CLK\\_ECKR\)](#) is cleared (HSE off)
- The AUX bit is set to indicate that the HSI/8 auxiliary clock source is forced.

You can clear the CSSD bit by software but the AUX bit is cleared only by reset.

To select a faster clock speed, you can modify the HSIDIV[1:0] bits in the CLK\_CKDIVR register after the CSSD bit in the CLK\_CSSR register is cleared.

If HSE is not the current clock master when a failure is detected, the master clock is not switched to the auxiliary clock and none of the above actions are performed except:

- The HSEEN bit is cleared in the CLK\_ECKR register, HSE is then switched OFF
- The CSSD bit is set in the CLK\_CSSR register and interrupt is generated if CSSDIE is also set, it can be cleared by software.

If HSE is not the current clock master and the master clock switch to HSE is ongoing, the SWBSY bit in the CLK\_SWCR register must be cleared by software before clearing the CSSD bit.

If HSE is selected by CCOSEL to be in output mode (see [Clock-out capability \(CCO\)](#)) when a failure is detected, the selection is automatically changed to force HSI (HSIDIV) instead of HSE.

## 9.7 Clock-out capability (CCO)

The configurable Clock Output (CCO) capability allows you to output a clock on the external CCO pin. You can select one of 6 clock signals as CCO clock:

- $f_{HSE}$
- $f_{HSI}$
- $f_{HSIDIV}$
- $f_{LSI}$
- **$f_{MASTER}$**
- $f_{CPU}$  (with current prescaling selection)

Note:

**50% duty cycle is not guaranteed on all possible prescaled values**

**The selection is controlled by the CCOSEL[3:0] bits in the Configurable clock output register (CLK\_CCOR).**

The user has to select first the desired clock for the dedicated I/O pin (see Pin Description chapter). This I/O must be set at 1 in the corresponding Px\_CR1 register to be set as input with pull-up or push-pull output.

**The sequence to really output the chosen clock starts with CCOEN=1 in Configurable clock output register (CLK\_CCOR).**

The CCOBSY is set to indicate that the configurable clock output system is operating. As long as the CCOBSY bit is set, the CCOSEL bits are write protected.

The CCO automatically activates the target oscillator if needed. The CCORDY bit is set when the chosen clock is ready.

To disable the clock output the user has to clear the CCOEN bit. Both CCOBSY and CCORDY remain at 1 till the shut down is completed. The time between the clear of CCOEN and the reset of the two flags can be relatively long, for instance in case the selected clock output is very slow compared to  $f_{CPU}$ .

## 9.8 CLK interrupts

The following interrupts can be generated by the clock controller:

- Master clock source switch event
- Clock Security System event

Both interrupts are individually maskable.

**Table 16. CLK interrupt requests**

Interrupt event	Event flag	Enable control bit	Exit from wait	Exit from Halt
CSS event	CSSD	CSSDIE	Yes	No
Master clock switch event	SWIF	SWIEN	Yes	No

## 9.9 CLK register description

### 9.9.1 Internal clock register (CLK\_ICKR)

Address offset: 0x00

Reset value: 0x01

7	6	5	4	3	2	1	0
Reserved	REGAH	LSIRDY	LSIEN	FHW	HSIRDY	HSIEN	
r	rw	r	rw	rw	r	rw	

Bits 7:6 Reserved, must be kept cleared.

Bit 5 **REGAH**: Regulator power off in Active-halt mode

This bit is set and cleared by software. When it is set, the main voltage regulator is powered off as soon as the MCU enters Active-halt mode, so the wakeup time is longer.

- 0: MVR regulator ON in Active-halt mode
- 1: MVR regulator OFF in Active-halt mode

Bit 4 **LSIRDY**: Low speed internal oscillator ready

This bit is set and cleared by hardware.

- 0: LSI clock not ready
- 1: LSI clock ready

Bit 3 **LSIEN**: Low speed internal RC oscillator enable

This bit is set and cleared by software. It is set by hardware whenever the LSI oscillator is required, for example:

- When switching to the LSI clock (see CLK\_SWR register)
- When LSI is selected as the active CCO source (see CLK\_CCOR register)
- When BEEP is enabled (BEEPEN bit set in the BEEP\_CSR register)
- When LSI measurement is enabled (MSR bit set in the AWU\_CSR register)

It cannot be cleared when LSI is selected as master clock source (CLK\_CMSR register), as active CCO source or as clock source for the AWU peripheral or independent Watchdog.

0: Low-speed internal RC off

1: Low-speed internal RC on

Bit 2 **FHWU**: Fast wakeup from Halt/Active-halt modes

This bit is set and cleared by software.

0: Fast wakeup from Halt/Active-halt modes disabled

1: Fast wakeup from Halt/Active-halt modes enabled

Bit 1 **HSIRDY**: High speed internal oscillator ready

This bit is set and cleared by hardware.

0: HSI clock not ready

1: HSI clock ready

Bit 0 **HSIEN**: High speed internal RC oscillator enable

This bit is set and cleared by software. It is set by hardware whenever the HSI oscillator is required, for example:

- When activated as safe oscillator by the CSS
- When switching to HSI clock (see CLK\_SWR register)
- When HSI is selected as the active CCO source (see CLK\_CCOR register)

It cannot be cleared when HSI is selected as clock master (CLK\_CMSR register), as active CCO source or if the safe oscillator (AUX) is enabled.

0: High-speed internal RC off

1: High-speed internal RC on

## 9.9.2 External clock register (CLK\_ECKR)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						HSERDY	HSEEN
r						r	rw

Bits 7:2 Reserved, must be kept cleared.

Bit 1 **HSERDY**: High speed external crystal oscillator ready

This bit is set and cleared by hardware.

0: HSE clock not ready

1: HSE clock ready (HSE clock is stabilized and available)

Bit 0 **HSEEN**: High speed external crystal oscillator enable

This bit is set and cleared by software. It can be used to switch the external crystal oscillator on or off. It is set by hardware in the following cases:

- When switching to HSE clock (see CLK\_SWR register)
- When HSE is selected as the active CCO source (see CLK\_CCOR register)

It cannot be cleared when HSE is selected as clock master (indicated in CLK\_CMSR register) or as the active CCO source.

0: HSE clock off

1: HSE clock on

### 9.9.3 Clock master status register (CLK\_CMSR)

Address offset: 0x03

Reset value: 0xE1

7	6	5	4	3	2	1	0
CKM[7:0]							
r	r	r	r	r	r	r	r

#### Bits 7:0 CKM[7:0]: Clock master status bits

These bits are set and cleared by hardware. They indicate the currently selected master clock source. An invalid value occurring in this register will automatically generate an MCU reset.

0xE1: HSI selected as master clock source (reset value)

0xD2: LSI selected as master clock source (only if LSI\_EN option bit is set)

0xB4: HSE selected as master clock source

### 9.9.4 Clock master switch register (CLK\_SWR)

Address offset: 0x04

Reset value: 0xE1

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

#### Bits 7:0 SWI[7:0]: Clock master selection bits

These bits are written by software to select the master clock source. Its contents are write protected while a clock switch is ongoing (while the SWBSY bit is set). They are set to the reset value (HSI) if the AUX bit is set in the CLK\_CSSR register. If Fast Halt wakeup mode is selected (FHW bit =1 in CLK\_ICKR register) then these bits are set by hardware to E1h (HSI selected) when resuming from Halt/Active-halt mode.

→ 0xE1: HSI selected as master clock source (reset value) ↗

→ 0xD2: LSI selected as master clock source (only if LSI\_EN option bit is set)

→ 0xB4: HSE selected as master clock source

### 9.9.5 Switch control register (CLK\_SWCR)

Address offset: 0x05

Reset value: 0XX

7	6	5	4	3	2	1	0
Reserved				SWIF	SWIEN	SWEN	SWBSY
	r			rc_w0	rw	rw	rw

Bits 7:4 Reserved.

Bit 3 SWIF: Clock switch interrupt flag

This bit is set by hardware and cleared by software writing 0. Its meaning depends on the status of the SWEN bit. Refer to [Figure 22](#) and [Figure 23](#).

" In manual switching mode (SWEN = 0):

- 0: Target clock source not ready
- 1: Target clock source ready

" In automatic switching mode (SWEN = 1):

- 0: No clock switch event occurred
- 1: Clock switch event occurred

Bit 2 SWIEN: Clock switch interrupt enable

This bit is set and cleared by software.

- 0: Clock switch interrupt disabled
- 1: Clock switch interrupt enabled

→ Bit 1 SWEN: Switch start/stop

This bit is set and cleared by software. Writing a 1 to this bit enables switching the master clock to the source defined in the CLK\_SWR register.

- 0: Disable clock switch execution
- 1: Enable clock switch execution

→ Bit 0 SWBSY: Switch busy

This bit is set and cleared by hardware. It can be cleared by software to reset the clock switch process.

- 0: No clock switch ongoing
- 1: Clock switch ongoing

### 9.9.6 Clock divider register (CLK\_CKDIVR)

Address offset: 0x06

Reset value: 0x18

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

Bits 7:5 Reserved, must be kept cleared.

Bits 4:3 **HSIDIV[1:0]**: High speed internal clock prescaler

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

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

Bits 2:0 **CPUDIV[2:0]**: CPU clock prescaler

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

- 000:  $f_{CPU} = f_{MASTER}$
- 001:  $f_{CPU} = f_{MASTER}/2$
- 010:  $f_{CPU} = f_{MASTER}/4$
- 011:  $f_{CPU} = f_{MASTER}/8$
- 100:  $f_{CPU} = f_{MASTER}/16$
- 101:  $f_{CPU} = f_{MASTER}/32$
- 110:  $f_{CPU} = f_{MASTER}/64$
- 111:  $f_{CPU} = f_{MASTER}/128$

### 9.9.7 Peripheral clock gating register 1 (CLK\_PCKENR1)

Address offset: 0x07

Reset value: 0xFF

7	6	5	4	3	2	1	0
PCKEN1[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **PCKEN1[7:0]**: Peripheral clock enable

These bits are written by software to enable or disable the  $f_{MASTER}$  clock to the corresponding peripheral (see [Table 17](#)).

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

**Table 17. Peripheral clock gating bits**

Control bit	Peripheral
PCKEN17	TIM1
PCKEN16	TIM3
PCKEN15	TIM2/TIM5 (product dependent)
PCKEN14	TIM4/ TIM6 (product dependent)
PCKEN13	UART1/2/3/4 (product dependent, see datasheet for bit assignment table)
PCKEN12	
PCKEN11	SPI
PCKEN10	I <sup>2</sup> C

### 9.9.8 Peripheral clock gating register 2 (CLK\_PCKENR2)

Address offset: 0x0A

Reset value: 0xFF

7	6	5	4	3	2	1	0
PCKEN2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **PCKEN2[7:0]**: Peripheral clock enable

These bits are written by software to enable or disable the  $f_{MASTER}$  clock to the corresponding peripheral. See [Table 17](#)

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

**Table 18. Peripheral clock gating bits**

Control bit	Peripheral
PCKEN27	CAN (product dependent, see datasheet)
PCKEN26	Reserved
PCKEN25	Reserved
PCKEN24	Reserved
PCKEN23	ADC
PCKEN22	AWU
PCKEN21	Reserved
PCKEN20	Reserved

### 9.9.9 Clock security system register (CLK\_CSSR)

Address offset: 0x08

Reset value: 0x00

7	6	5	4	3	2	1	0
		Reserved		CSSD	CSSDIE	AUX	CSSEN
		r		rc_w0	rw	r	rwo

Bits 7:4 Reserved, must be kept cleared.

Bit 3 **CSSD**: Clock security system detection

This bit is set by hardware and cleared by software writing 0.

0: CSS is off or no HSE crystal clock disturbance detected.

1: HSE crystal clock disturbance detected.

Bit 2 **CSSDIE**: Clock security system detection interrupt enable

This bit is set and cleared by software.

0: Clock security system interrupt disabled

1: Clock security system interrupt enabled

Bit 1 **AUX**: Auxiliary oscillator connected to master clock

This bit is set and cleared by hardware.

0: Auxiliary oscillator is off.

1: Auxiliary oscillator (HSI/8) is on and selected as current clock master source.

Bit 0 **CSSEN**: Clock security system enable

This bit can be read many times and be written once-only by software.

0: Clock security system off

1: Clock security system on

### 9.9.10 Configurable clock output register (CLK\_CCOR)

Address offset: 0x09

Reset value: 0x00

7	6	5	4	3	CCOSEL[3:0]	2	1	0
Reserved	CCOBSY	CCORDY			CCOSEL[3:0]			CCOEN
r	r	r	rw	rw	rw	rw	rw	rw

Bit 7 Reserved, must be kept cleared.

Bit 6 **CCOBSY**: Configurable clock output busy

This bit is set and cleared by hardware. It indicates that the selected CCO clock source is being switched-on and stabilized. While CCOBSY is set, the CCOSEL bits are write-protected. CCOBSY remains set until the CCO clock is enabled.

0: CCO clock not busy

1: CCO clock busy

Bit 5 **CCORDY**: Configurable clock output ready

This bit is set and cleared by hardware. It indicates that the CCO clock is being output.

0: CCO clock not available

1: CCO clock available

Bits 4:1 **CCOSEL[3:0]**: Configurable clock output selection.

These bits are written by software to select the source of the output clock available on the CLK\_CCO pin. They are write-protected when CCOBSY is set.

0000:  $f_{HSDIV}$

0001:  $f_{LSI}$

0010:  $f_{HSE}$

0011: Reserved

0100:  $f_{CPU}$

0101:  $f_{CPU}/2$

0110:  $f_{CPU}/4$

0111:  $f_{CPU}/8$

1000:  $f_{CPU}/16$

1001:  $f_{CPU}/32$

1010:  $f_{CPU}/64$

1011:  $f_{HSI}$

1100:  $f_{MASTER}$

1101:  $f_{CPU}$

1110:  $f_{CPU}$

1111:  $f_{CPU}$

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

### 9.9.11 HSI clock calibration trimming register (CLK\_HSITRIMR)

Address offset: 0x0C

Reset value: 0x00

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

Bits 7:4 Reserved, must be kept cleared.

Bits 3:0 **HSITRIM[3:0]** HSI trimming value

These bits are written by software to fine tune the HSI calibration.

*Note: In high density STM8S and STM8AF devices, only bits 2:0 are available.*

*In other devices, bits 3:0 are available to achieve a better HSI resolution. Compatibility with bits 2:0 can be selected through options bytes (refer to datasheet).*

### 9.9.12 SWIM clock control register (CLK\_SWIMCCR)

Address offset: 0x0D

Reset value: 0bXXXX XXX0

7	6	5	4	3	2	1	0
Reserved				SWIMCLK			
	r						rw

Bits 7:1 Reserved.

Bit 0 **SWIMCLK** SWIM clock divider

This bit is set and cleared by software.

0: SWIM clock divided by 2

1: SWIM clock not divided by 2

## 9.10 CLK register map and reset values

Table 19. CLK register map and reset values

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	CLK_ICKR Reset value	-0	-0	REGAH0	LSIRDY0	LSIEN0	FHWU0	HSIRDY0	HSIEN1
0x01	CLK_ECKR Reset value	-0	-0	-0	-0	-0	-0	HSERDY0	HSEEN0
0x02	Reserved area (1 byte)								
0x03	CLK_CMSR Reset value	CKM71	CKM61	CKM51	CKM40	CKM30	CKM20	CKM10	CKM01
0x04	CLK_SWR Reset value	SWI71	SWI61	SWI51	SWI40	SWI30	SWI20	SWI10	SWI01
0x05	CLK_SWCR Reset value	-x	-x	-x	-x	SWIFx	SWIENx	SWENx	SWBSYx
0x06	CLK_CKDIVR Reset value	-0	-0	-0	HSIDIV11	HSIDIV01	CPUDIV20	CPUDIV120	CPUDIV00
0x07	CLK_PCKENR1 Reset value	PCKEN171	PCKEN161	PCKEN151	PCKEN141	PCKEN131	PCKEN121	PCKEN111	PCKEN101
0x08	CLK_CSSR Reset value	-0	-0	-0	-0	CSSD0	CSSDIE0	AUX0	CSSEN0
0x09	CLK_CCOR Reset value	-0	CCOBSY0	CCORDY0	CCOSEL30	CCOSEL20	CCOSEL10	CCOSEL00	CCOEN0
0x0A	CLK_PCKENR2 Reset value	PCKEN271	PCKEN261	PCKEN251	PCKEN241	PCKEN231	PCKEN221	PCKEN211	PCKEN201
0x0B	Reserved area (1 byte)								
0x0C	CLK_HSITRIMR Reset value	-0	-0	-0	-0	-0	HSITRIM20	HSITRIM10	HSITRIM00
0x0D	CLK_SWIMCCR Reset value	-x	-x	-x	-x	-x	-x	-x	SWIMCLK0

## 10 Power management

By default, after a system or power reset, the microcontroller is in Run mode. In this mode the CPU is clocked by  $f_{CPU}$  and executes the program code, the system clocks are distributed to the active peripherals and the microcontroller is drawing full power.

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 clocks
- 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
- Active-halt (configurable for slow or fast wakeup)
- Halt (configurable for slow or fast wakeup)

You can select one of these three modes and configure them to obtain the best compromise between lowest power consumption, fastest start-up time and available wakeup sources.

### 10.1 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 leakages. This is so small, it is only significant in Halt and Active-halt modes (refer to [Section 10.2: 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
- Analog performance
- 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.

### 10.1.1 Clock management for low consumption

#### Slowing down the system clock

In Run mode, choosing the oscillator to be used as the system clock source is very important to ensure the best compromise between performance and consumption. The selection is done by programming the clock controller registers. Refer to the [Clock control \(CLK\)](#) section.

As a further measure,  $f_{CPU}$  can be reduced by writing to the CPUDIV[2:0] bits in the [Clock divider register \(CLK\\_CKDIVR\)](#). This reduces the speed of the CPU and consequently the power consumption of the MCU. The other peripherals (clocked by  $f_{MASTER}$ ) are not affected by this setting.

To return to full speed at any time in Run mode, clear the CPUDIV[2:0] bits.

#### 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 the [Clock control \(CLK\)](#) section.

These settings are effective in both Run and Wait modes.

## 10.2 Low power modes

The main characteristics of the four low power modes are summarized in [Table 20](#).

**Table 20. Low power mode management**

Mode (consumption level)	Main voltage regulator	Oscillators	CPU	Peripherals	Wakeup trigger event
Wait (-)	On	On	Off	On <sup>(1)</sup>	All internal interrupts (including AWU) or external interrupts, reset
Active-halt (---)	On	Off except LSI (or HSE)	Off	Only AWU <sup>(2)</sup>	AWU or external <sup>(3)</sup> interrupts, reset
Active-halt with MVR auto power off (----)	Off (low power regulator on)	Off except LSI only	Off	Only AWU <sup>(2)</sup>	AWU or external <sup>(3)</sup> interrupts, reset
Halt (-----)	Off (low power regulator on)	Off	Off	Off <sup>(2)</sup>	External <sup>(3)</sup> interrupts, reset

1. If the peripheral clock is not disabled by peripheral clock gating function.
2. If activated, BEEP or IWDG stay switched on. In this case, the LSI clock is forced to run.
3. Including communication peripheral interrupts.

### 10.2.1 Wait mode

Wait mode is entered from Run mode by executing a WFI (wait for interrupt) instruction: this stops the CPU but allows the other peripherals and interrupt controller to continue to run. Therefore the consumption decreases accordingly. Wait mode can be combined with PCG (peripheral clock gating), reduced CPU clock frequency and low mode clock source selection (LSI, HSI) to further reduce the power consumption of the device. Refer to the [Clock control \(CLK\)](#) description.

In Wait mode, all the registers and RAM contents are preserved, the previously defined clock configuration remains unchanged ([Clock master status register \(CLK\\_CMSR\)](#)).

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

### 10.2.2 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.

In Halt mode, all the registers and RAM contents are preserved, by default the clock configuration remains unchanged ([Clock master status register \(CLK\\_CMSR\)](#)).

The MCU enters Halt mode when a HALT instruction is executed. Wakeup from Halt mode is triggered by an external interrupt, sourced by a GPIO port configured as interrupt input or an Alternate Function pin capable of triggering a peripheral interrupt.

In this mode the MVR regulator is switched off to save power. Only the LPVR regulator (and brown-out reset) is active.

#### Fast clock wakeup

The HSI RC start-up time is much faster than the HSE crystal start-up time (refer to the Electrical Parameters in the datasheet). Therefore, to optimize the MCU wakeup time, it is recommended to select the HSI clock as the  $f_{MASTER}$  clock source before entering Halt mode.

This selection can be done without clock switching using the FHWU bit in the [Internal clock register \(CLK\\_ICKR\)](#). Refer to the [Clock control \(CLK\)](#) chapter.

### 10.2.3 Active-halt modes

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

In Active-halt mode, the main oscillator, the CPU and almost all the peripherals are stopped.

Only the LSI RC or HSE oscillators are running to drive the AWU counters and IWD counter if enabled.

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

### Main voltage regulator (MVR) auto power-off

By default the main voltage regulator is kept on Active-halt mode. Keeping it active ensures fast wakeup from Active-halt mode. However, the current consumption of the MVR is non-negligible.

To further reduce current consumption, the MVR regulator can be powered off automatically when the MCU enters Active-halt mode. To configure this feature, set the REGAH bit in the [Internal clock register \(CLK\\_ICKR\)](#) register. In this mode:

- The MCU core is powered only by the LPVR regulator (same as in Halt mode).
- Only the LSI clock source can be used, as the HSE clock current consumption is too high for the LPVR.

The Main voltage regulator is powered on again at wakeup and it requires a longer wakeup time (refer to the datasheet electrical characteristics section for wakeup timing and current consumption data).

### Fast clock wakeup

As described for Halt mode, in order to get the shortest wakeup time, it is recommended to select HSI as the  $f_{MASTER}$  clock source. The FHWU bit is also available to save switching time.

A fast wakeup time is very important in Active-halt mode. It supplements the effect of CPU processing performance by helping to minimize the time the MCU stays in Run mode between two periods in low power mode, and thus reduces the overall average power consumption.

Since the clock is not automatically switched to the original master clock, the application must restore the clock source before entering Halt/Active-halt mode as soon as it is ready.

## 10.3 Additional analog power controls

### 10.3.1 Fast Flash wakeup from Halt mode

By default the Flash is in power-down state when the microcontroller enters Halt mode. The current leakage is negligible, resulting in very low consumption in Halt mode. However the Flash wakeup time is relatively slow (several  $\mu$ s).

If you need the application to wakeup quickly from Halt mode, set the HALT bit in [Section 4.8.1: Flash control register 1 \(FLASH\\_CR1\)](#). This ensures that the Flash is in Standby mode when the microcontroller enters in Halt mode. Its wakeup time is reduced to a few ns. However, in this case the consumption is increased up to several  $\mu$ As.

Refer to the electrical characteristics section of the datasheet for more details.

### 10.3.2 Very low Flash consumption in Active-halt mode

By default, in Active-halt mode, the Flash remains in operating mode to ensure the fastest wakeup time, however in this case the power consumption is not optimized.

To optimize the power consumption you can set the AHALT bit in Flash control register 1 (FLASH\_CR1). This will switch the Flash to power-down state when entering Active-halt mode. The consumption decreases but the wakeup time increases up to a few  $\mu$ s.

## 11 General purpose I/O ports (GPIO)

### 11.1 Introduction

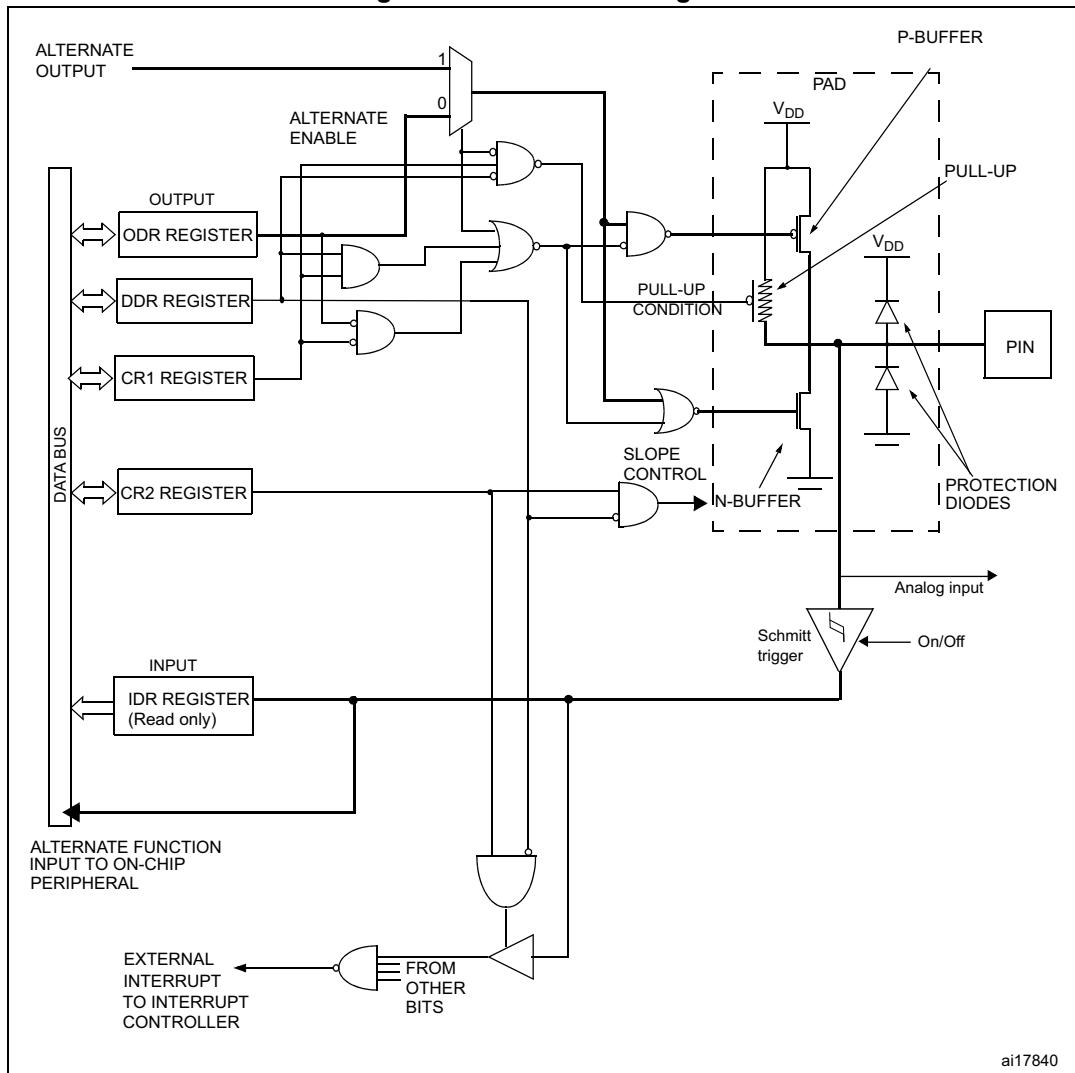
General purpose input/output ports are used for data transfers between the chip and the external world. An I/O port can contain up to eight pins. Each pin can be individually programmed as a digital input or digital output. In addition, some ports may have alternate functions like analog inputs, external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

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.

### 11.2 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
- Input Schmitt trigger can be disabled on analog inputs for reduced power consumption
- Read-modify-write possible on data output latch
- 5 V-tolerant inputs
- I/O state guaranteed in voltage range 1.6 V to  $V_{DDIOMax}$

Figure 24. GPIO block diagram



Note: The output stage is disabled when the analog input is selected.

## 11.3 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 21](#).

**Table 21. I/O port configuration summary**

Mode	DDR bit	CR1 bit	CR2 bit	Function	Pull-up	P-buffer	Diodes	
							to V <sub>DD</sub>	to V <sub>SS</sub>
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	On	On
	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<sub>DD</sub> is not implemented in true open drain pads. A local protection between the pad and V<sub>OL</sub> is implemented to protect the device against positive stress.

### 11.3.1 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 11.7: Input mode details on page 109](#) for information on analog input, external interrupts and Schmitt trigger enable/disable.

As shown in [Table 21](#), 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.

### 11.3.2 Output modes

Setting the DDR<sub>x</sub> 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 11.8: Output mode details on page 110](#) for more information.

## 11.4 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.

## 11.5 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<sub>DD</sub> or V<sub>SS</sub> 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 a factory setting (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'.

## 11.6 Low power modes

**Table 22. 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.

**Note:** If PA1/PA2 pins are used to connect an external oscillator, to ensure a lowest power consumption in Halt mode, PA1 and PA2 must be configured as input pull-up.

## 11.7 Input mode details

### 11.7.1 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.

### 11.7.2 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.

### 11.7.3 Analog channels

Analog I/O channels can be selected by the ADC peripheral. The corresponding input and output stages are then automatically disabled. As mentioned in the next section, the input Schmitt trigger should be disabled in the ADC\_TDR register when using the analog channels.

**Table 23. Recommended and non-recommended configurations for analog input**

DDR	CR1	CR2	ADC_TDR	Configuration	Comments
0	0	0	1	Floating Input without interrupt, Schmitt trigger disabled	Recommended analog input configuration
0	1	x	x	Input with pull-up enabled	Not recommended for analog input when analog voltage is present since these configurations cause excess current flow on the input pin.
1	0	x	x	Output	
1	1	x	x	Output	Both input and output stages are disabled on ADC selected channel.

#### 11.7.4 Schmitt trigger

On all I/Os with an analog input, it is possible to disable the Schmitt trigger, even if the corresponding ADC channel is not enabled. The two registers ADC\_TDRH and ADC\_TDRL allow to disable the Schmitt trigger.

Setting one bit in these registers leads to disabling the corresponding Schmitt trigger input buffer.

In case an I/O is used as analog input, and the corresponding ADC channel is enabled (CH[3:0] bits in ADC\_CSR register), the Schmitt trigger is disabled, whatever the status of the corresponding bit in ADC\_TDRH or ADC\_TDRL registers.

#### 11.7.5 Analog function

Selected I/Os can be used to deliver analog signal to ADC, Comparators or DAC periphery. The GPIO pin have to be configured in the input floating configuration without interrupt (default state) to use it for analog function. The current consumption of the IO with enabled analog function can be reduced by disabling unused Schmitt trigger in IO input section either by ADC\_TRIGRx register in ADC interface (see [Section 14.3.15: Schmitt trigger disabling](#)) or by switching on a corresponding analog switch in RI by setting corresponding CHxE bit in RI\_IOSRx (see [Section 11.2.2: I/O groups](#)). See the product datasheet for pins with analog functions.

### 11.8 Output mode details

#### 11.8.1 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 output pins must be set-up as push-pull, fast slope for optimal operation.

#### 11.8.2 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.

## 11.9 GPIO registers

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

### 11.9.1 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							

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.

### 11.9.2 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.

### 11.9.3 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							

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

### 11.9.4 Port x control register 1 (Px\_CR1)

Address offset: 0x03

Reset value: 0x00 except for PD\_CR1 which reset value is 0x02.

7	6	5	4	3	2	1	0
C17	C16	C15	C14	C13	C12	C11	C10
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 [Table 21](#)).

– **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).*

### 11.9.5 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							

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

### 11.9.6 GPIO register map and reset values

Each GPIO port has five registers mapped as shown in [Table 24](#). 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 24. 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 <sup>(1)</sup> 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

1. PD\_CR reset value is 0x02.

## 12 Auto-wakeup (AWU)

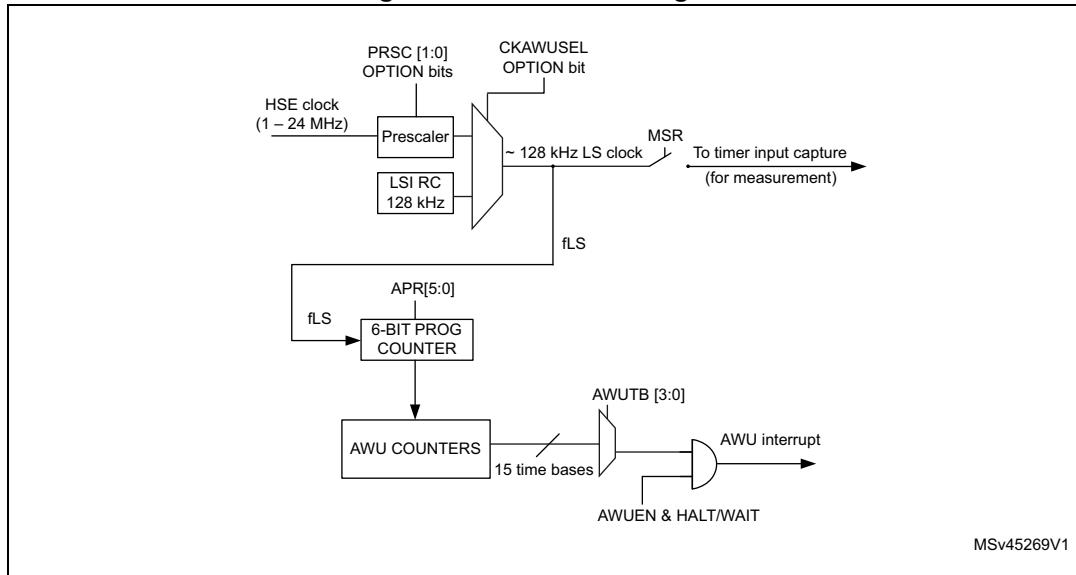
### 12.1 AWU introduction

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 or the HSE crystal oscillator clock divided by a prescaler.

### 12.2 LSI clock measurement

To ensure the best possible accuracy when using the LSI clock, its frequency can be measured with TIM3 or TIM1 timer input capture 1 (see datasheet for information on which timer is connected in the specific product).

**Figure 25. AWU block diagram**



1. The LS clock source is selected by programming the CKAWUSEL option bit as explained in the clock controller chapter.

## 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 TIM3 or TIM1 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 ( $\text{APR}_{\text{DIV}}$ ).

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

**Table 25. Time base calculation table**

Interval range		AWUTB[3:0]	$\text{APR}_{\text{DIV}}$ formula for time interval calculation	$\text{APR}_{\text{DIV}}$ range
$f_{LS} = f$	$f_{LS} = 128\text{kHz}$			
$2/f - 64/f$	0.015625 ms - 0.5 ms	0001	$\text{APR}_{\text{DIV}}/f_{LS}$	2 to 64
$2 \times 32/f - 2 \times 2 \times 32/f$	0.5 ms - 1.0 ms	0010	$2 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2 \times 64/f - 2 \times 2 \times 64/f$	1 ms - 2 ms	0011	$2^2 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^2 \times 64/f - 2^2 \times 128/f$	2 ms - 4 ms	0100	$2^3 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^3 \times 64/f - 2^3 \times 128/f$	4 ms - 8 ms	0101	$2^4 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^4 \times 64/f - 2^4 \times 128/f$	8 ms - 16 ms	0110	$2^5 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^5 \times 64/f - 2^5 \times 128/f$	16 ms - 32 ms	0111	$2^6 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^6 \times 64/f - 2^6 \times 128/f$	32 ms - 64 ms	1000	$2^7 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^7 \times 64/f - 2^7 \times 128/f$	64 ms - 128 ms	1001	$2^8 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^8 \times 64/f - 2^8 \times 128/f$	128 ms - 256 ms	1010	$2^9 \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^9 \times 64/f - 2^9 \times 128/f$	256 ms - 512 ms	1011	$2^{10} \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^{10} \times 64/f - 2^{10} \times 128/f$	512 ms - 1.024 s	1100	$2^{11} \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^{11} \times 64/f - 2^{11} \times 128/f$	1.024 s - 2.048 s	1101	$2^{12} \times \text{APR}_{\text{DIV}}/f_{LS}$	32 to 64
$2^{11} \times 130/f - 2^{11} \times 320/f$	2.080 s - 5.120 s	1110	$5 \times 2^{11} \times \text{APR}_{\text{DIV}}/f_{LS}$	26 to 64
$2^{11} \times 330/f - 2^{12} \times 960/f$	5.280 s - 30.720 s	1111	$30 \times 2^{11} \times \text{APR}_{\text{DIV}}/f_{LS}$	11 to 64

In order to obtain the right values for AWUTB[3:0] and  $\text{APR}_{\text{DIV}}$ , you have to:

- First, search the interval range corresponding to the desired time interval. This gives the AWUTB[3:0] value.
- Then  $\text{APR}_{\text{DIV}}$  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  $2^{11} \times 128/f_{LS}$  and  $2^{11} \times 130/f_{LS}$  or between  $2^{11} \times 320/f_{LS}$  and  $2^{11} \times 330/f_{LS}$ , the value closer to the target one must be chosen.

**Example 1**

- $f_{LS} = 128 \text{ kHz}$
- Target time interval = 6 ms

The appropriate interval range is: 4 ms - 8 ms  
so the AWUTB[3:0] value is 0x5.

The APR<sub>DIV</sub> value is:

$$6 \text{ ms} = 2^4 \times \text{APR}_{\text{DIV}} / f_{LS} \Rightarrow \text{APR}_{\text{DIV}} = (6 \times 10^{-3} \times f_{LS}) / 2^4 = 48$$

so the APR[5:0] value is 48 (0x30)

**Example 2**

- $f_{LS} = 128 \text{ kHz}$
- Target time interval = 3 s

The appropriate interval range is: 2.080 s - 5.120 s  
So the AWUTB[3:0] value is 0xE.

The APR<sub>DIV</sub> value is:

$$3 \text{ s} = 5 \times 2^{11} \times \text{APR}_{\text{DIV}} / f_{LS} \Rightarrow \text{APR}_{\text{DIV}} = (3 \times f_{LS}) / (5 \times 2^{11}) = 37.5$$

So the AWUTB[3:0] can be either 37 or 38 which gives a time base of 2.96s or 3.04s respectively. This is not exactly 3s.

### 12.3.3 LSI clock frequency measurement

The frequency dispersion of the low speed internal RC (LSI) oscillator after RC factory trimming is 128 kHz +/- 12.5% on the whole temperature range. 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
r	rc_r	rw		r			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 LS 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]			
r				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]			
r				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:  $2 \times APR_{DIV}/f_{LS}$

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

0100:  $2^3 APR_{DIV}/f_{LS}$

0101:  $2^4 APR_{DIV}/f_{LS}$

0110:  $2^5 APR_{DIV}/f_{LS}$

0111:  $2^6 APR_{DIV}/f_{LS}$

1000:  $2^7 APR_{DIV}/f_{LS}$

1001:  $2^8 APR_{DIV}/f_{LS}$

1010:  $2^9 APR_{DIV}/f_{LS}$

1011:  $2^{10} APR_{DIV}/f_{LS}$

1100:  $2^{11} APR_{DIV}/f_{LS}$

1101:  $2^{12} APR_{DIV}/f_{LS}$

1110:  $5 \times 2^{11} APR_{DIV}/f_{LS}$

1111:  $30 \times 2^{11} APR_{DIV}/f_{LS}$

#### 12.4.4 AWU register map and reset values

Table 26. AWU register map

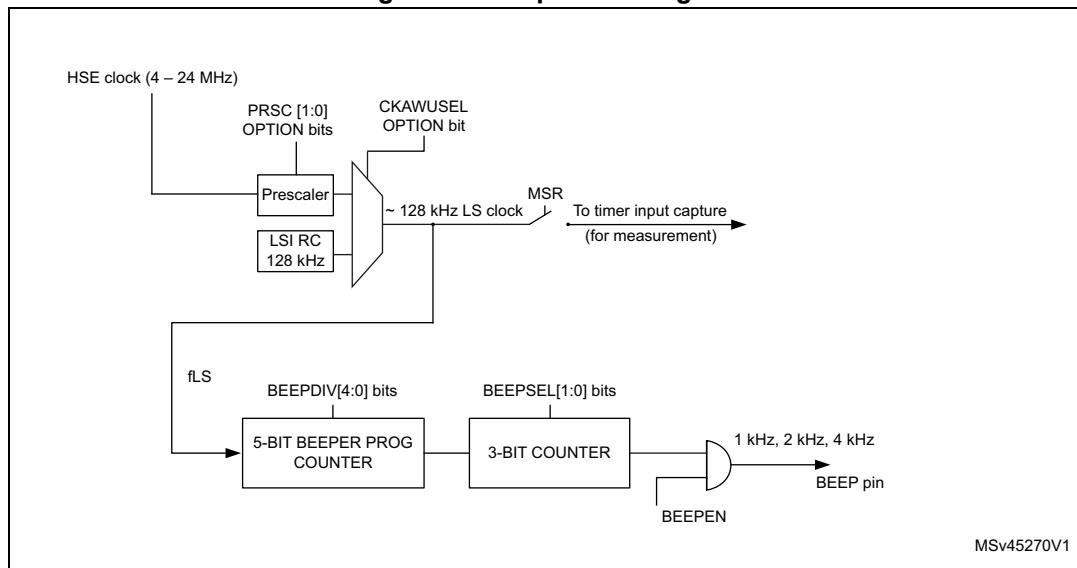
Address offset	Register name	7	6	5	4	3	2	1	0
0x00	AWU_CSR Reset value	-0	-0	AWUF 0	AWUEN 0	-0	-0	-0	MSR 0
0x01	AWU(APR Reset value	-0	-0	APR5 1	APR4 1	APR3 1	APR2 1	APR1 1	APR0 1
0x02	AWU_TBR Reset value	-0	-0	-0	-0	AWUTB3 0	AWUTB2 0	AWUTB1 0	AWUTB0 0

## 13 Beeper (BEEP)

### 13.1 Beeper 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 128 kHz.

**Figure 26. Beep block diagram**



MSv45270V1

### 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 128 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  $\text{BEEP}_{\text{DIV}}$  value as follows, where A and x are the integer and fractional part of  $f_{\text{LS}}/8$  (in kHz):
 
$$\text{BEEP}_{\text{DIV}} = A - 2 \text{ when } x \text{ is less than or equal to } A/(1+2^*A), \text{ else}$$

$$\text{BEEP}_{\text{DIV}} = A - 1$$
3. Write the resulting  $\text{BEEP}_{\text{DIV}}$  value in the  $\text{BEEPDIV}[4:0]$  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:  $f_{\text{LS}}/(8 \times \text{BEEP}_{\text{DIV}})$  kHz output
- 01:  $f_{\text{LS}}/(4 \times \text{BEEP}_{\text{DIV}})$  kHz output
- 1x:  $f_{\text{LS}}/(2 \times \text{BEEP}_{\text{DIV}})$  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  $\text{BEEP}_{\text{DIV}}$ .

0x00:  $\text{BEEP}_{\text{DIV}} = 2$

0x01:  $\text{BEEP}_{\text{DIV}} = 3$

...

0x0E:  $\text{BEEP}_{\text{DIV}} = 16$

0x0F:  $\text{BEEP}_{\text{DIV}} = 17$

0x1E:  $\text{BEEP}_{\text{DIV}} = 32$

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

### 13.3.2 Beeper register map and reset values

Table 27. 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 Independent watchdog (IWDG)

### 14.1 IWDG introduction

The independent watchdog peripheral can be used to resolve processor malfunctions due to hardware or software failures. It is clocked by the 128 kHz LSI internal RC clock source, and thus stays active even if the main clock fails.

### 14.2 IWDG functional description

*Figure 27* shows the functional blocks of the independent watchdog module.

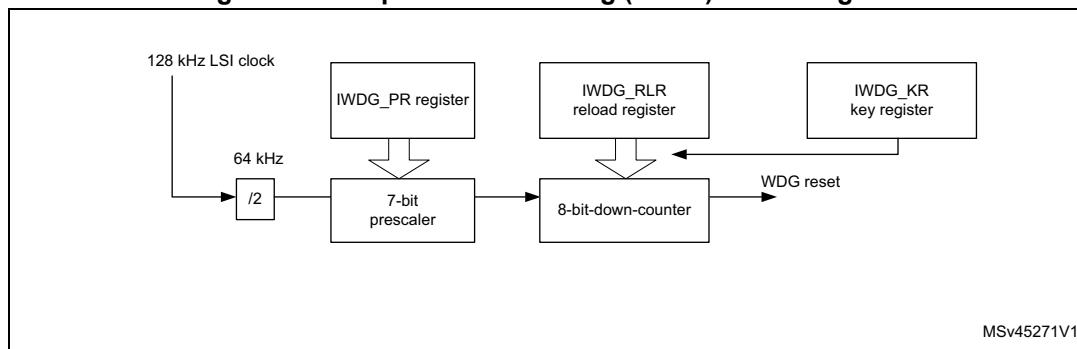
When the independent watchdog is started by writing the value 0xCC in the key register (IWDG\_KR), the counter starts counting down from the reset value of 0xFF. When it reaches the end of count value (0x00) a reset signal is generated (IWDG RESET).

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 14.3: IWDG registers](#) for details on the IWDG registers.

**Figure 27. Independent watchdog (IWDG) 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 timeout period can be configured through the IWDG\_PR and IWDG\_RLR registers. It is determined by the following equation:

$$T = 2 \times T_{LSI} \times P \times R$$

where:

$T$  = 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:

$$D = T + 6 \times T_{LSI}$$

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

**Table 28. Watchdog timeout period (LSI clock frequency = 128 kHz)**

Prescaler divider	PR[2:0] bits	Timeout	
		RL[7:0]= 0x00	RL[7:0]= 0xFF
/4	0	62.5 µs	15.90 ms
/8	1	125 µs	31.90 ms
/16	2	250 µs	63.70 ms
/32	3	500 µs	127 ms
/64	4	1.00 ms	255 ms
/128	5	2.00 ms	510 ms
/256	6	4.00 ms	1.02 s

## 14.3 IWDG registers

### 14.3.1 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 14.2](#)).

### 14.3.2 Prescaler register (IWDG\_PR)

Address offset: 0x01

Reset value: 0x00

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

Bits 7:3 Reserved

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

These bits are write access protected (see [Section 14.2](#)). 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

### 14.3.3 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 14.2](#)). 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 28](#).

### 14.3.4 IWDG register map and reset values

Table 29. 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] 1111111			

## 15 Window watchdog (WWDG)

### 15.1 WWDG introduction

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

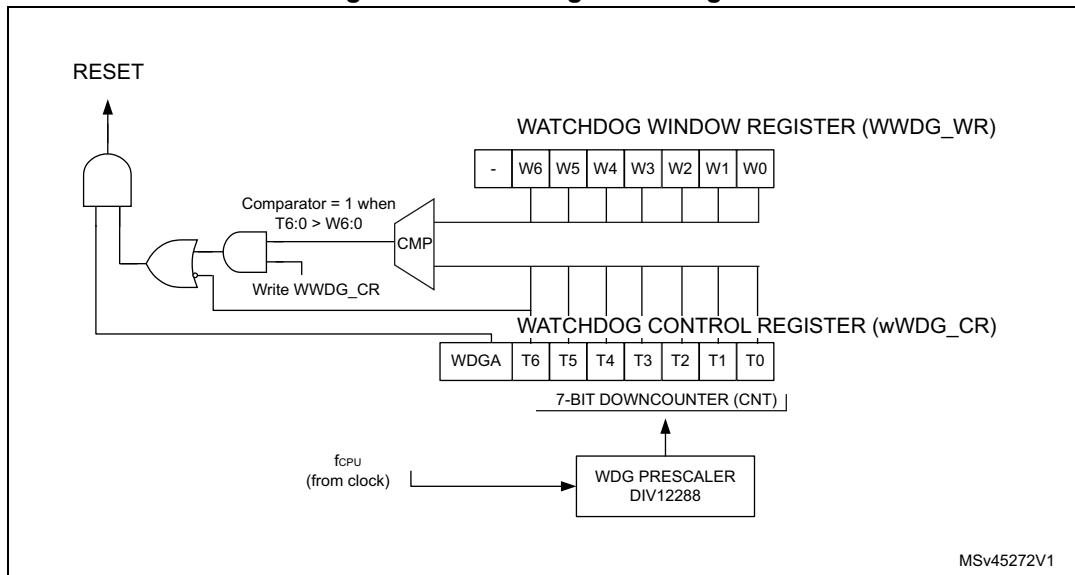
### 15.2 WWDG main features

- 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 30](#))
- Hardware/software watchdog activation (selectable by option byte)
- Optional reset on HALT instruction (configurable by option byte)

### 15.3 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 28. 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 (see [Figure 29](#)):

- 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 (see [Figure 29: Approximate timeout duration](#)).

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 30](#)).

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 30](#) 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.

## 15.4 How to program the watchdog timeout

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

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

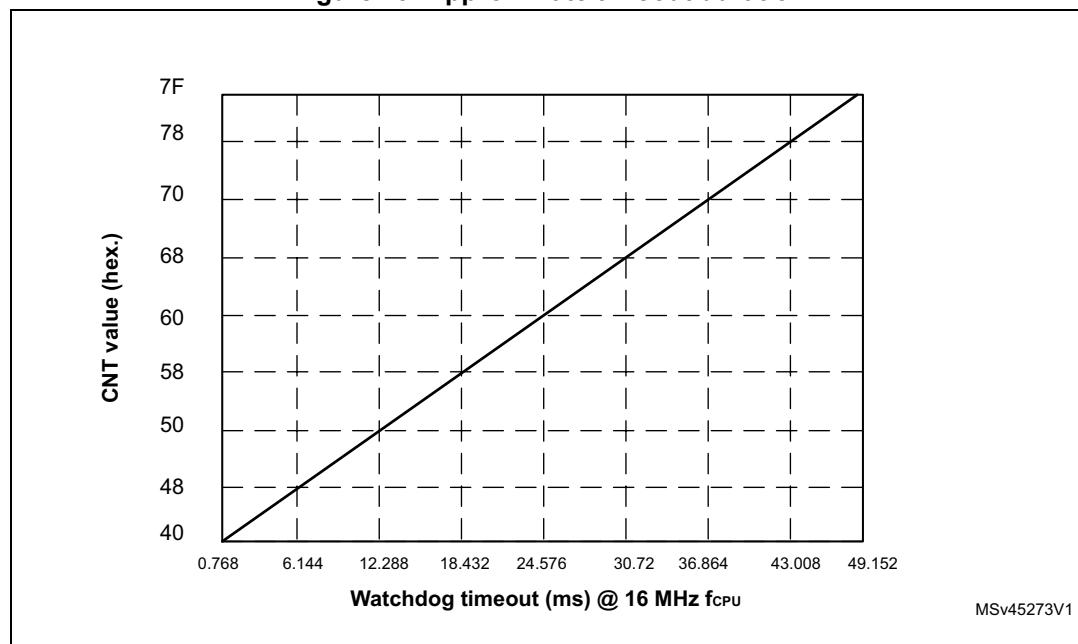
where TCPU is the peripheral clock period expressed in ms

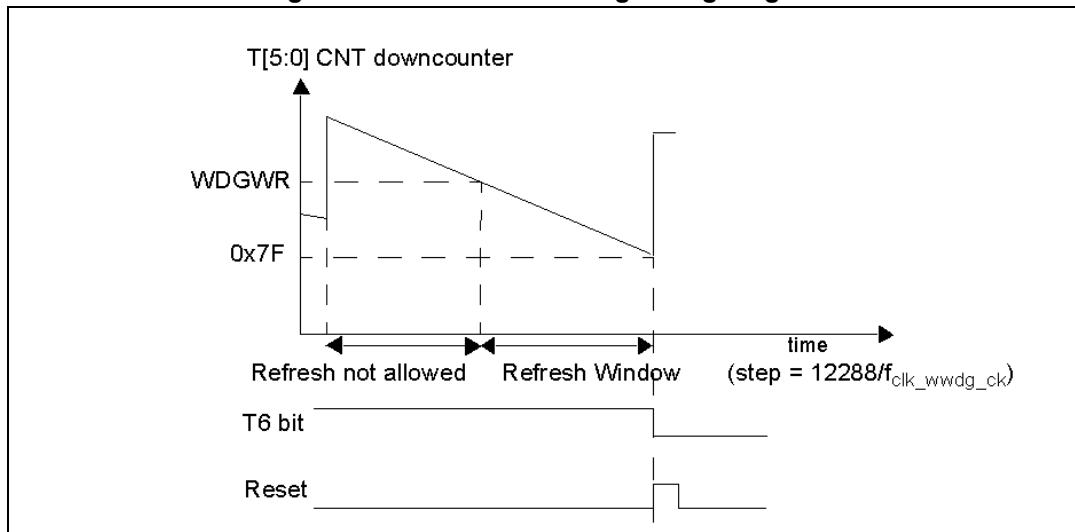
---

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

---

Figure 29. Approximate timeout duration



**Figure 30. Window watchdog timing diagram****Table 30. Window watchdog timing example**

T[6:0]	f <sub>CPU</sub> (MHz)	
	2	16
40h	6.144	0.768
7Fh	393.216	49.152

## 15.5 WWDG low power modes

**Table 31. Effect of low power modes on WWDG**

Mode	Description	
Wait	No effect on watchdog: The downcounter continues to decrement.	
WWDG_HALT in option byte		
Halt	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 <a href="#">Section 15.8</a> 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.

## 15.6 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.

## 15.7 Using Halt mode with the WWDG (WWDGHALT option)

The following recommendation applies if Halt mode is used when the watchdog is enabled.

Before executing the HALT instruction, refresh the WDG counter, to avoid an unexpected WWDG reset immediately after waking up the microcontroller.

## 15.8 WWDG interrupts

None.

## 15.9 WWDG registers

### 15.9.1 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						

Bit 7 **WDGA**: Activation bit<sup>(1)</sup>

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<sub>CPU</sub> 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.

### 15.9.2 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							

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.

## 15.10 Window watchdog register map and reset values

Table 32. 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

## 16 Timer overview

The devices in the STM8S and STM8AF family may be equipped with up to three different timer types: Advanced control (TIM1), general purpose (TIM2/TIM3/TIM5), and basic timers (TIM4/TIM6). The timers share the same architecture, but some have additional unique features. The common timer architecture, which includes identical register mapping and common basic features, simplifies their use and makes it easier to design applications. [Table 33](#) shows the main timer characteristics.

In STM8S and STM8AF devices with TIM1, TIM5 and TIM6, the timers do not share any resources but they can be linked together and synchronized as described in [Synchronization between TIM1, TIM5 and TIM6 timers on page 159](#). In STM8S and STM8AF devices with TIM1, TIM2, TIM3 and TIM4, the timers are not linked together.

This section gives a comparison of the different timer features ([Table 34](#)), a glossary of internal timer signal names ([Table 35](#)).

[Section 17: 16-bit advanced control timer \(TIM1\)](#) contains a full description of all the various timer modes. The other timer sections ([Section 18](#) and [Section 19](#)) are more brief and give only specific details on each timer, its block diagram, and register description.

**Table 33. 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 34. Timer feature comparison

Timer	Counter resolution	Counter type	Prescaler factor	Capture/compare channels	Complementary outputs	Repetition counter	External trigger input	External break input	Timer synchronization/chaining
TIM1 (advanced control timer)	16-bit	Up/down	Any integer from 1 to 65536	4	3	Yes	1	1	With TIM5/TIM6
TIM2 (general purpose timer)		Up	Any power of 2 from 1 to 32768	3	None	No	0	0	No
TIM3 (general purpose timer)				2					
TIM4 (basic timer)	8-bit	Up	Any power of 2 from 1 to 128	0	None	No	1 (shared with TIM1)	0	Yes
TIM5 (general purpose timer)	16-bit		Any power of 2 from 1 to 32768	3					
TIM6 (basic timer)	8-bit		Any power of 2 from 1 to 128	0					

## 16.2 Glossary of timer signal names

Table 35. Glossary of internal timer signals

Internal signal name	Description	Related figures
BI	Break interrupt	<a href="#">Figure 31: TIM1 general block diagram on page 140</a>
CC <i>i</i> : CC1I, CC2I, CC3I, CC4I	Capture/compare interrupt	<a href="#">Figure 31: TIM1 general block diagram on page 140</a>
CK_CNT	Counter clock	<a href="#">Figure 35: Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2 on page 144</a>
CK_PSC	Prescaler clock	<a href="#">Figure 35: Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2 on page 144</a>
CNT_EN	Counter enable	<a href="#">Figure 35: Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2 on page 144</a>
CNT_INIT	Counter initialize	<a href="#">Figure 45: TI2 external clock connection example on page 152</a>
ETR	External trigger from TIMx_ETR pin	<a href="#">Figure 47: External trigger input block diagram on page 154</a>
ETRF	External trigger filtered	
ETRP	External trigger prescaled	
fMASTER	Timer peripheral clock from clock controller (CLK)	<a href="#">Figure 20: Clock tree and Figure 12: Clock structure on page 61</a>
IC <i>i</i> : IC1, IC2	Input capture	<a href="#">Figure 64: Input stage of TIM 1 channel 1 on page 168</a>
IC <i>i</i> PS: IC1PS, IC2PS	Input capture prescaled	
MATCH1	Compare match	<a href="#">Figure 54: Trigger/master mode selection blocks on page 161 and Section 17.7.2: Control register 2 (TIM1_CR2) on page 190</a>
OC <i>i</i> : OC1, OC2	Timer output channel	<a href="#">Figure 68: Detailed output stage of channel with complementary output (channel 1) on page 172</a>
OC <i>i</i> REF: OC1REF, OC2REF	Output compare reference signal	
TGI	Trigger interrupt	<a href="#">Figure 43: Clock/trigger controller block diagram on page 151</a>
TI <i>i</i> : TI1, TI2	Timer input	<a href="#">Figure 64: Input stage of TIM 1 channel 1 on page 168</a>
TI <i>i</i> F: TI1F, TI2F	Timer input filtered	
TI1_ED	Timer input edge detector	
TI <i>i</i> FPn: TI1FP1, TI1FP2, TI2FP1, TI2FP2, TI3FP3, TI3FP4, TI4FP3, TI4FP4	Timer input filtered prescaled	
TRC	Trigger capture	<a href="#">Figure 44: Control circuit in normal mode, fCK_PSC = fMASTER on page 152</a>
TRGI	Trigger input to clock/trigger/slave mode controller	

**Table 35. Glossary of internal timer signals (continued)**

Internal signal name	Description	Related figures
UEV	Update event	<i>Figure 35: Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2 on page 144</i>
UIF	Update interrupt	

**Table 36. Explanation of indices 'i', 'n', and 'x'<sup>(1)</sup>**

i	Signal number: May be 1, 2, 3, 4 depending on the device
	Bit number: May be 1, 2, 3, 4 ..... depending on the device
	Register number: May be 1, 2, 3, 4 depending on the device
n	Signal number (when i is already used): May be 1, 2, 3, 4 depending on the device
x	Timer number: May be 1, 2, 3, 4, 5, 6 depending on the device
	Don't care (for bits)

1. These indices are used in [Section 17](#), [Section 18](#), and [Section 19](#).

## 17 16-bit advanced control timer (TIM1)

This section gives a description of the full set of timer features.

### 17.1 TIM1 introduction

TIM1 consists of a 16-bit up-down auto-reload counter driven by a programmable prescaler.

The timer 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 TIM5/TIM6 timers or external signals (external clock, reset, trigger and enable).

This timer is ideally suited for a wide range of control applications, including those requiring center-aligned PWM capability with complementary outputs and deadtime insertion.

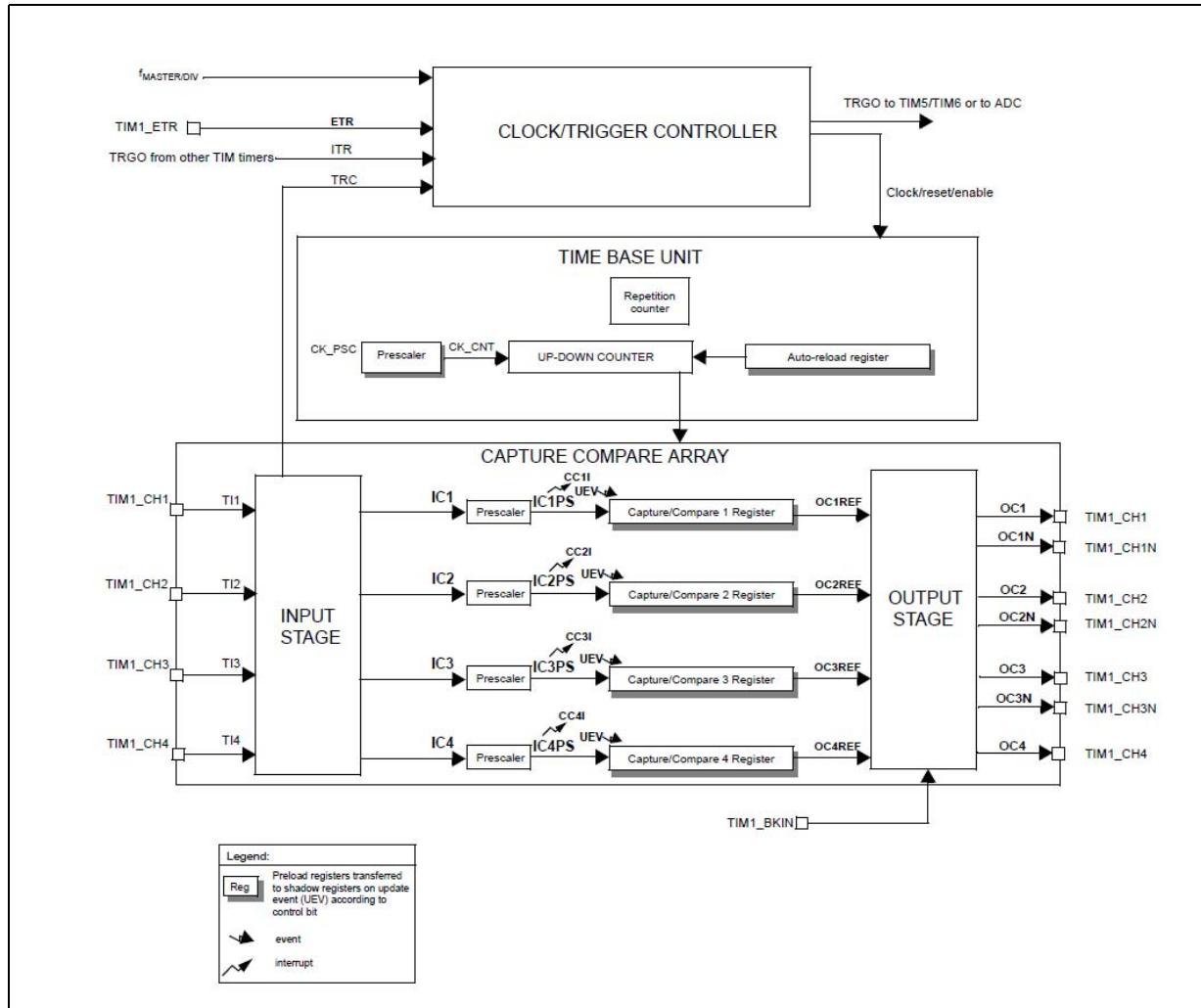
The timer clock can be sourced from internal clocks or from an external source selectable through a configuration register.

## 17.2 TIM1 main features

TIM1 features include:

- 16-bit up, down, up/down counter auto-reload counter
- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- 16-bit programmable prescaler allowing the counter clock frequency to be divided “on the fly” by any factor between 1 and 65536.
- Synchronization circuit to control the timer with external signals and to interconnect several timers (timer interconnection not implemented in some devices).
- 4 independent channels that can alternately be configured as:
  - Input capture
  - Output compare
  - PWM generation (edge and center-aligned mode)
  - 6-step PWM generation
  - One-pulse mode output
  - Complementary outputs on three channels with programmable deadtime insertion.
- Break input to put the timer output signals in reset state or in a known state.
- External trigger input pin (ETR)
- 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 31. TIM1 general block diagram

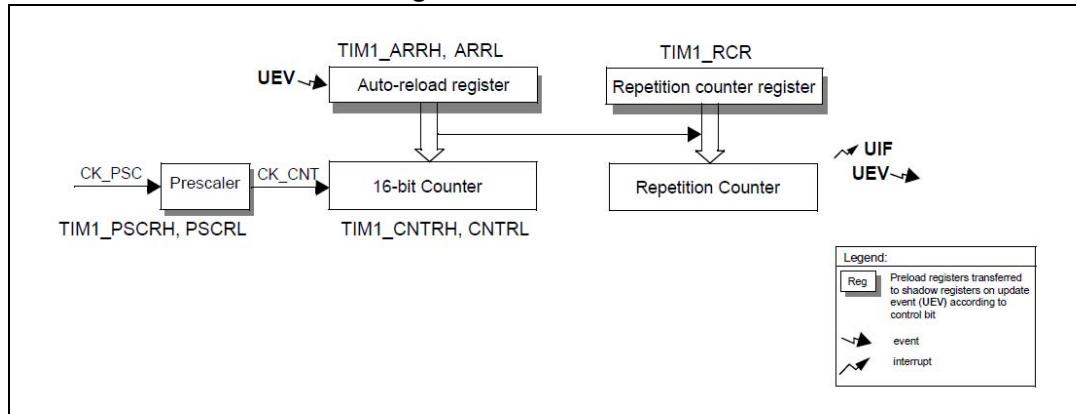


## 17.3 TIM1 time base unit

The timer has a time base unit that includes:

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

**Figure 32. Time base unit**



The 16-bit counter, the prescaler, the auto-reload register and the repetition counter 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 TIM1\_CR1 register). In this mode, when data is written to the auto-reload 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 TIM1\_CR1 register). In this mode, when data is written to the auto-reload 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 TIM1\_EGR register
- By a 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 (TIM1\_ARR) and the buffer of the prescaler is reloaded with the preload value (content of the TIM1\_PSCR register).

The UEV can be disabled by setting the UDIS bit in the TIM1\_CR1

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIM1\_CR1 register is set.

*Note:*

*The actual counter enable signal CNT\_EN is set 1 clock cycle after CEN.*

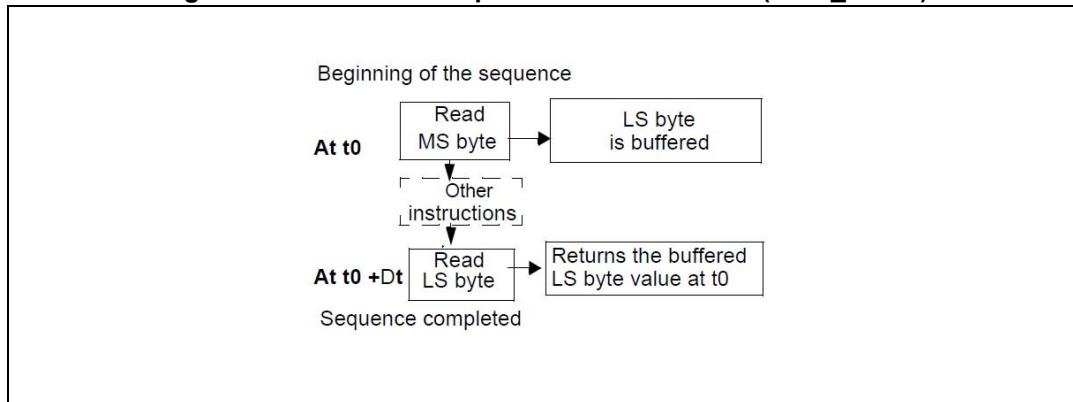
### 17.3.1 Reading and writing to the 16-bit counter

There is no buffering when writing to the counter. Both TIM1\_CNTRH and TIM1\_CNTL 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 an incorrect intermediate content.

An 8-bit buffer is implemented for the read. Software must read the MS byte first, after which the LS byte value is buffered automatically (see [Figure 33](#)). 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. It reads the LS byte first and returns an incorrect result.*

**Figure 33. 16-bit read sequence for the counter (TIM1\_CNTR)**



### 17.3.2 Write sequence for 16-bit TIM1\_ARR register

16-bit values are loaded in the TIM1\_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 which produces incorrect results.

### 17.3.3 Prescaler

The prescaler implementation is as follows:

The TIM1 prescaler is based on a 16-bit counter controlled through a 16-bit register (in TIM1\_PSCR register). It can be changed on the fly as this control register is buffered. It can divide the counter clock frequency by any factor between 1 and 65536.

The counter clock frequency is calculated as follows:

$$f_{CK\_CNT} = f_{CK\_PSC}/(PSCR[15:0]+1)$$

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.

To update the 16-bit prescaler, load two bytes in separate write operations starting with the MSB. Do not use the LDW instruction for this purpose as it writes the LSB first.

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

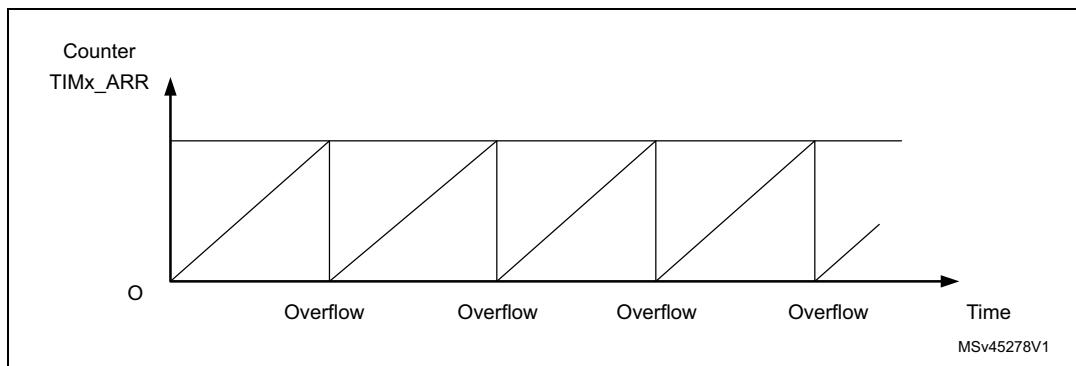
Read operations to the TIM1\_PSCR registers access the preload registers, so no special care needs to be taken to read them.

### 17.3.4 Up-counting mode

In up-counting mode, the counter counts from 0 to a user-defined compare value (content of the TIM1\_ARR register). It then restarts from 0 and generates a counter overflow event and a UEV if the UDIS bit is 0 in the TIM1\_CR1 register.

*Figure 34* shows an example of this counting mode.

**Figure 34. Counter in up-counting mode**



An update event can also be generated by setting the UG bit in the TIM1\_EGR register (either by software or by using the trigger controller).

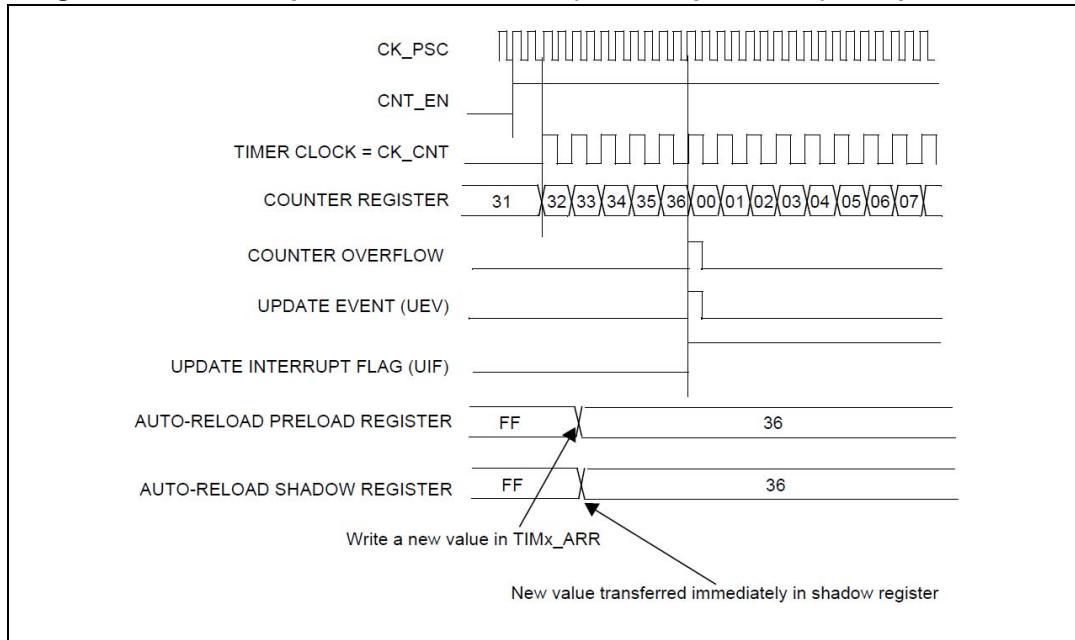
The UEV can be disabled by software by setting the UDIS bit in the TIM1\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. No UEV occurs until the UDIS bit has been written to 0. Note that the counter and the prescaler restart counting from 0 but, the prescaler division factor does not change. In addition, if the URS bit (update request selection) in the TIM1\_CR1 register is set, setting the UG bit generates an UEV without setting the UIF flag. Consequently, no interrupt request is sent. This avoids 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 TIM1\_SR1 register) is set (depending on the URS bit):

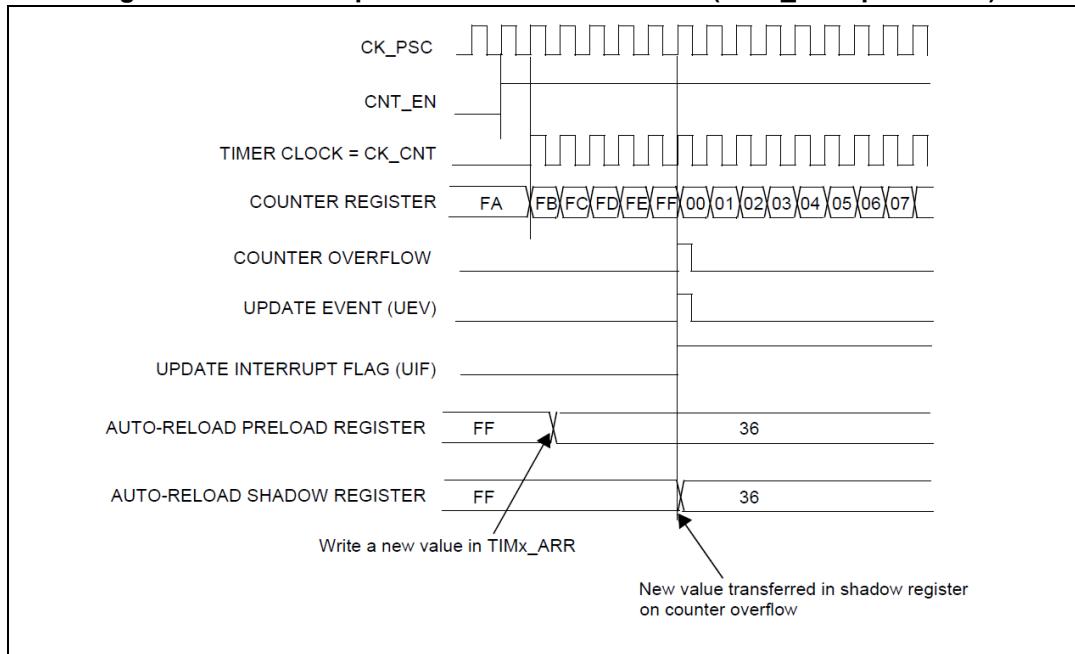
- The auto-reload shadow register is updated with the preload value (TIM1\_ARR).
- The buffer of the prescaler is reloaded with the preload value (content of the TIM1\_PSCR register).

*Figure 35* and *Figure 36* show two examples of the counter behavior for different clock frequencies when TIM1\_ARR = 0x36.

In *Figure 35*, the prescaler divider is set to 2, so the counter clock (CK\_CNT) frequency is at half the frequency of the prescaler clock source (CK\_PSC). The auto-reload preload is disabled (ARPE = 0). Consequently, the shadow register is immediately changed and counter overflow occurs when upcounting reaches 0x36. This generates a UEV.

**Figure 35. Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2**

In [Figure 36](#) the prescaler divider is set to 1, so CK\_CNT has the same frequency as CK\_PSC. The auto-reload preload is enabled (ARPE = 1), so the next counter overflow occurs at 0xFF. The new auto-reload value register value of 36h is taken into account after the overflow which generates a UEV.

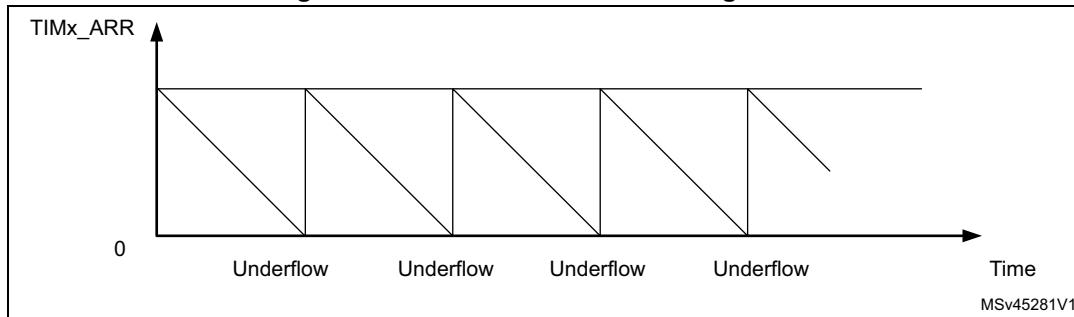
**Figure 36. Counter update event when ARPE = 1 (TIM1\_ARR preloaded)**

### 17.3.5 Down-counting mode

In down-counting mode, the counter counts from the auto-reload value (content of the TIM1\_ARR register) down to 0. It then restarts from the auto-reload value and generates a counter underflow and a UEV, if the UDIS bit is 0 in the TIM1\_CR1 register.

*Figure 37* shows an example of this counting mode.

**Figure 37. Counter in down-counting mode**



An update event can also be generated by setting the UG bit in the TIM1\_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 TIM1\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. No update event occurs until the 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 (without any change to the prescale rate).

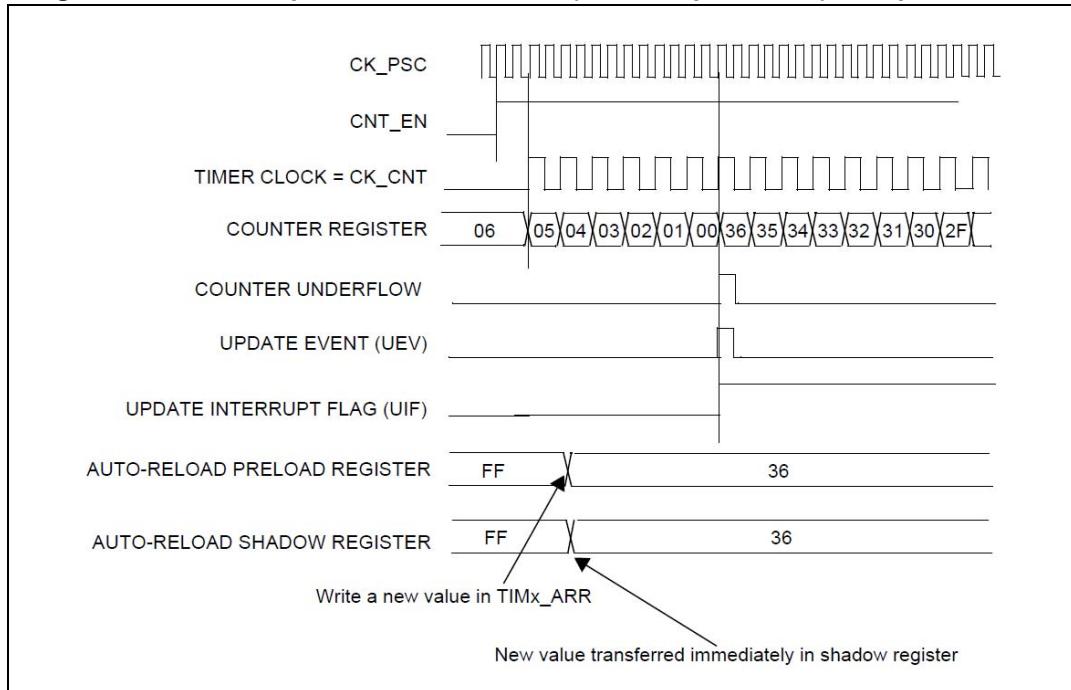
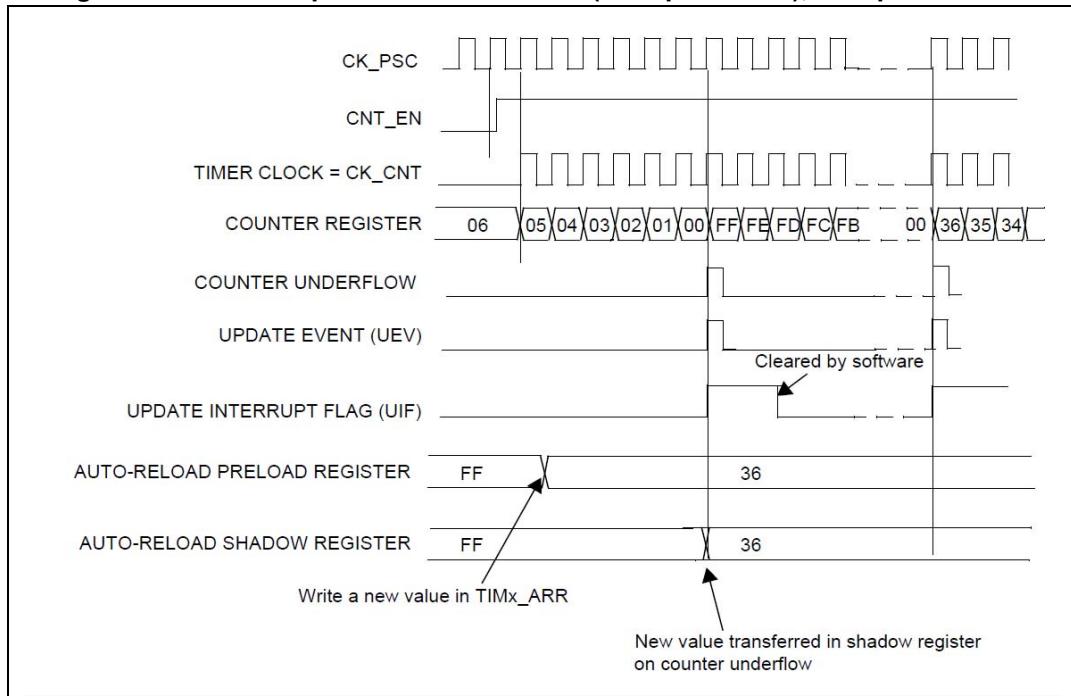
In addition, if the URS bit (update request selection) in the TIM1\_CR1 register is set, setting the UG bit generates a UEV without setting the UIF flag (thus no interrupt request is sent). This avoids generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIM1\_SR1 register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (content of the TIM1\_PSCR register),
- The auto-reload shadow register is updated with the preload value (content of the TIM1\_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

*Figure 38* and *Figure 39* show some examples of the counter behavior for different clock frequencies when TIM1\_ARR = 0x36.

In downcounting mode, preload is not normally used. Consequently, the new value is taken into account in the next period (see *Figure 38*).

**Figure 38. Counter update when ARPE = 0 (ARR not preloaded) with prescaler = 2****Figure 39. Counter update when ARPE = 1 (ARR preloaded), with prescaler = 1**

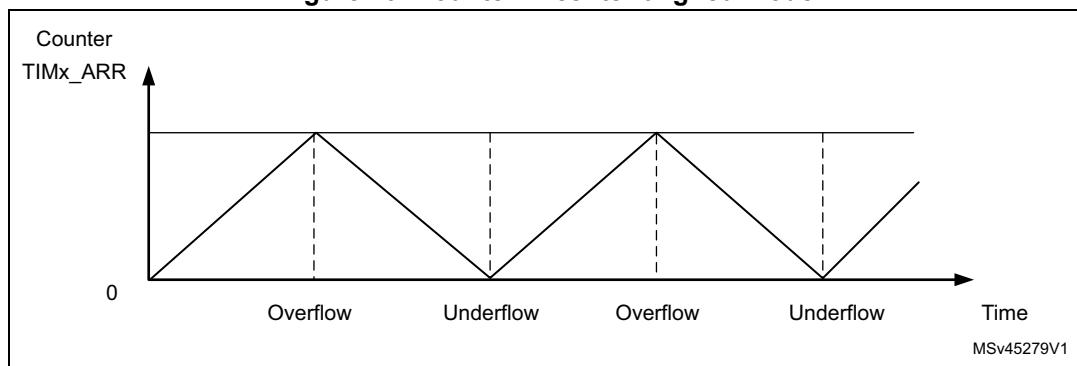
### 17.3.6 Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value of -1 (content of the TIM1\_ARR register). This generates a counter overflow event. The counter then counts down to 0 and generates a counter underflow event. After this, the counter restarts counting from 0.

In this mode, the direction bit (DIR) in the TIM1\_CR1 register cannot be written. It is updated by hardware and gives the current direction of the counter.

The [Figure 40](#) shows an example of this counting mode.

**Figure 40. Counter in center-aligned mode**



If the timer has a repetition counter (as in TIM1), the UEV is generated after up and down counting and repeated for the number of times programmed in the repetition counter register (TIM1\_RCR). Otherwise, the UEV is generated at each counter overflow and at each counter underflow.

Setting the UG bit in the TIM1\_EGR register (by software or by using the clock/trigger mode controller) also generates an update event. In this case, the counter and the prescaler restart counting from 0.

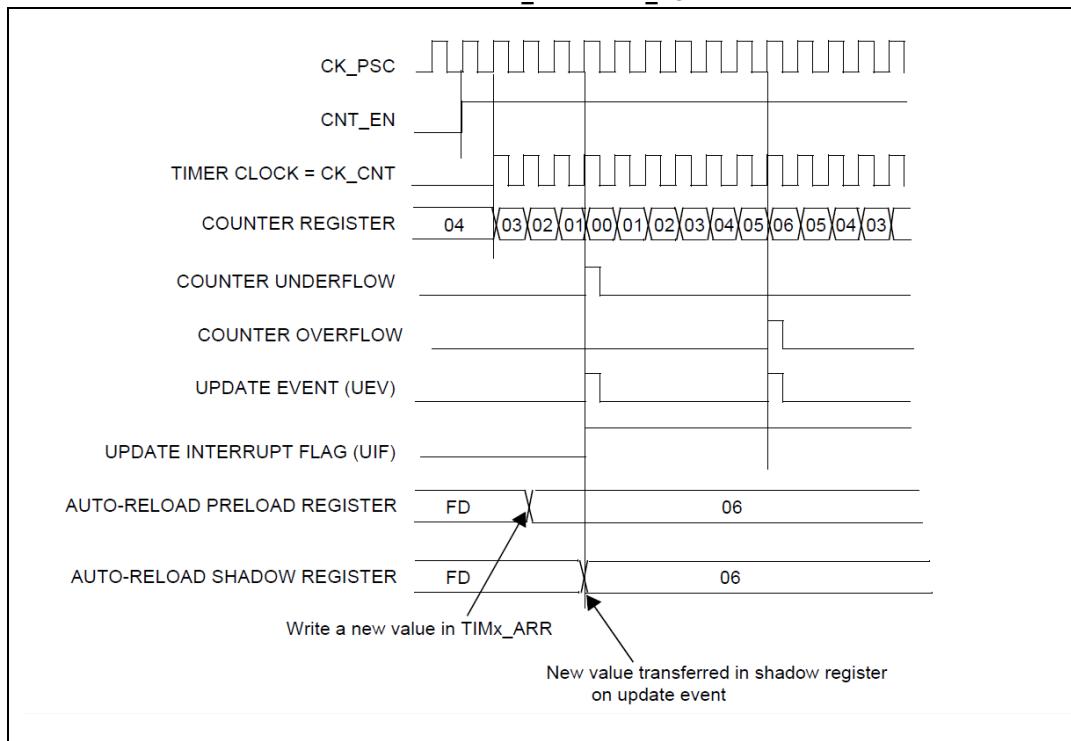
The UEV can be disabled by software by setting the UDIS bit in the TIM1\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. In this way, no update event occurs until the UDIS bit is written to 0. However, the counter continues counting up and down, based on the current auto-reload value. In timers with a repetition counter, the new update rate is used because the repetition register is not double buffered. For this reason, care must be taken when changing the update rate.

In addition, if the URS bit in the TIM1\_CR1 register is set, setting the UG bit generates a UEV without setting the UIF flag. Consequently, no interrupt request is sent. This avoids generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all registers are updated and the update flag (the UIF bit in the TIM1\_SR1 register) is set (depending on the URS bit).

- The buffer of the prescaler is reloaded with the preload value (content of the TIM1\_PSCR register).
- The auto-reload shadow register is updated with the preload value (content of the TIM1\_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).

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

Figure 41. Counter timing diagram,  $f_{CK\_CNT} = f_{CK\_PSC}$ ,  $TIM1\_ARR = 06h$ ,  $ARPE = 1$ 

Hints on using center-aligned mode:

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter starts counting up or down depending on the value written in the DIR bit in the TIM1\_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if a value is written in the counter that is greater than the auto-reload value ( $TIM1\_CNT > TIM1\_ARR$ ). For example, if the counter is counting up, it continues to do so.
  - The direction is updated if 0 or the  $TIM1\_ARR$  value are written in the counter but no UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIM1\_EGR register) just before starting the counter. Avoid writing to the counter while it is running.

### 17.3.7 Repetition down-counter

*Section 17.3: TIM1 time base unit* describes how the UEV is generated with respect to counter overflows/underflows. It is generated only when the repetition down-counter reaches zero. This can be useful while generating PWM signals.

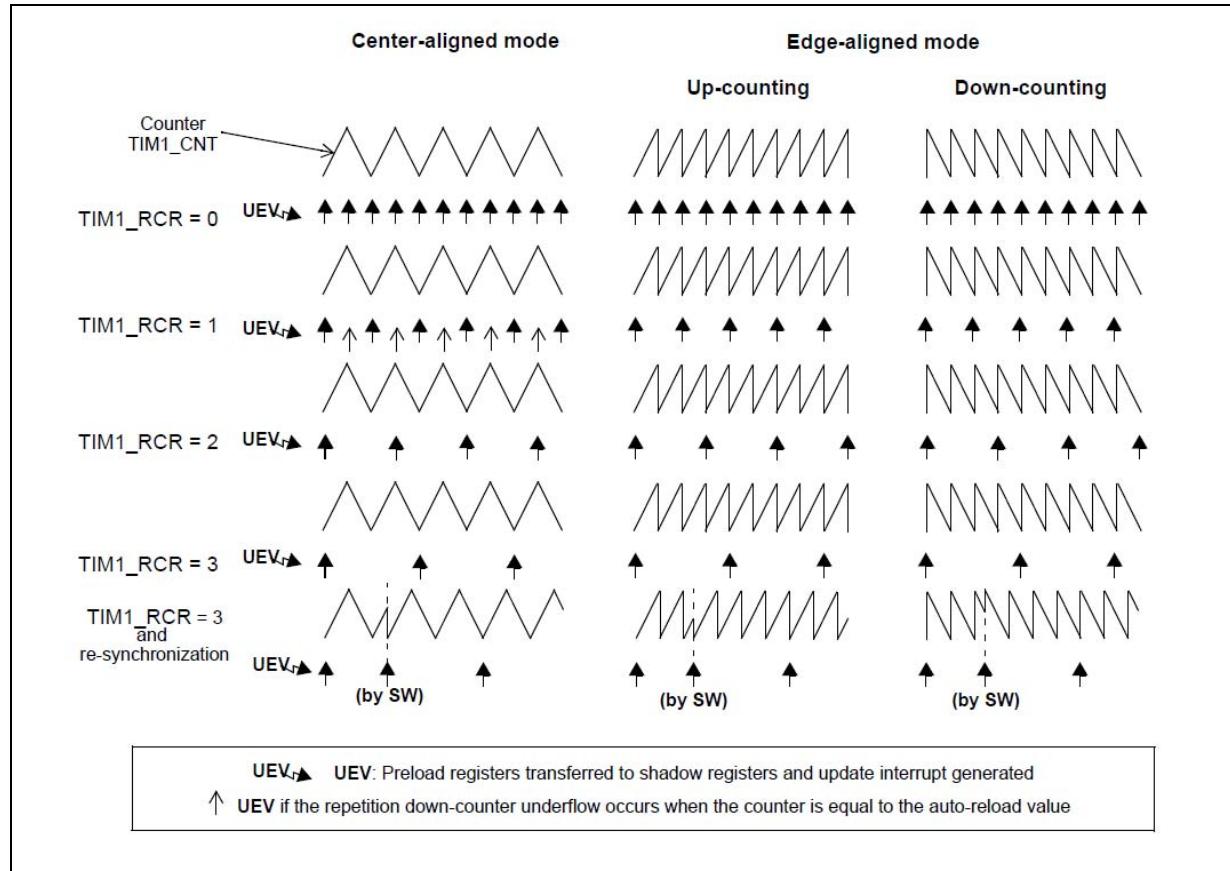
This means that data are transferred from the preload registers to the shadow registers (TIM1\_ARR auto-reload register, TIM1\_PSCR prescaler register, and TIM1\_CCRx capture/compare registers in compare mode) every 'n' counter overflow or underflow, where N is the value in the TIM1\_RCR repetition counter register.

The repetition down-counter is decremented:

- At each counter overflow in up-counting mode
- At each counter underflow in down-counting mode
- At each counter overflow and at each counter underflow in center-aligned mode.  
Although this limits the maximum number of repetitions to 128 PWM cycles, it makes it possible to update the duty cycle twice per PWM period. When refreshing compare registers only once per PWM period in center-aligned mode, maximum resolution is  $2 \times t_{CK\_PSC}$  due to the symmetry of the pattern.

The repetition down-counter is an auto-reload type, the repetition rate of which is maintained as defined by the TIM1\_RCR register value (refer to [Figure 42](#)). When the UEV is generated by software (by setting the UG bit in the TIM1\_EGR register) or by hardware (through the clock/trigger controller), it occurs immediately irrespective of the value of the repetition down-counter. The repetition down-counter is reloaded with the content of the TIM1\_RCR register.

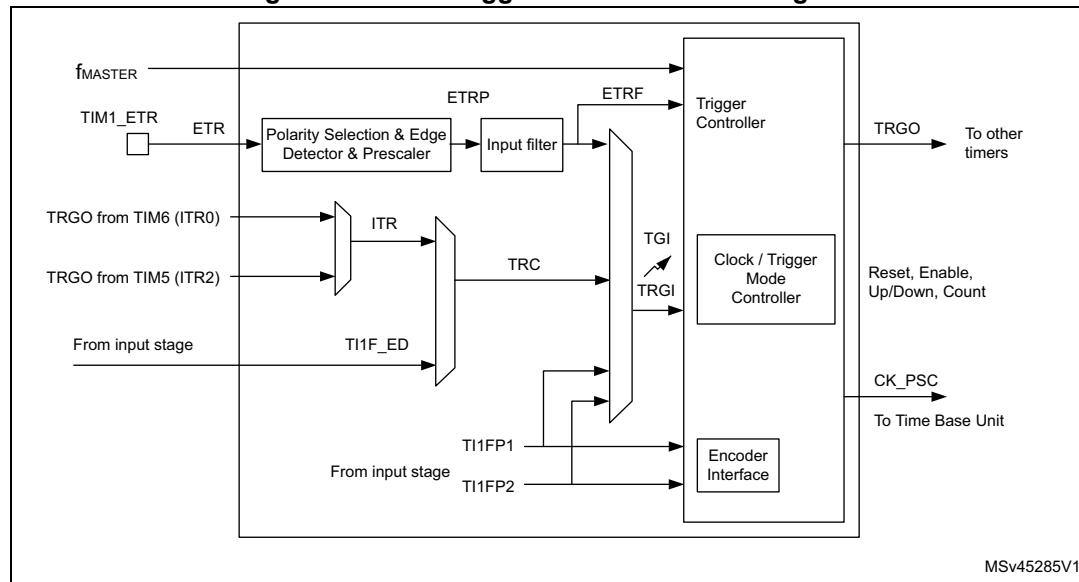
Figure 42. Update rate examples depending on mode and TIM1\_RCR register settings



## 17.4 TIM1 clock/trigger controller

The clock/trigger controller allows the timer clock sources, input triggers, and output triggers to be configured. The block diagram is shown in [Figure 43](#).

**Figure 43. 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:

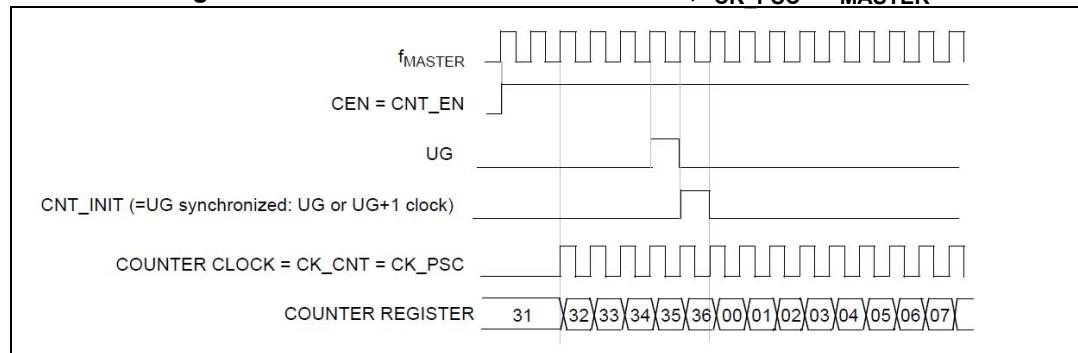
- Internal clock ( $f_{MASTER}$ )
- External clock mode 1: External timer input (Tlx)
- External clock mode 2: External trigger input (ETR)
- Internal trigger inputs (ITR): using one timer as prescaler for another timer. Refer to [Using one timer as prescaler for another timer on page 161](#) for more details.

### 17.4.2 Internal clock source ( $f_{MASTER}$ )

If both the clock/trigger mode controller and the external trigger input are disabled (SMS = 000 in TIM1\_SMCR and ECE = 0 in the TIM1\_ETR register), the CEN, DIR, and UG bits behave as 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.

The figure below shows the behavior of the control circuit and the up-counter in normal mode, without the prescaler.

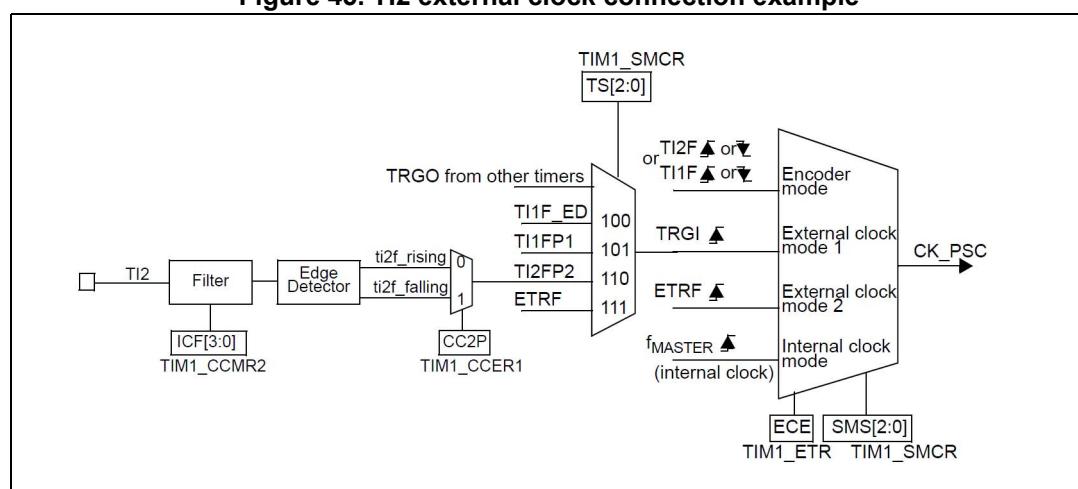
**Figure 44. Control circuit in normal mode,  $f_{CK\_PSC} = f_{MASTER}$**



### 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 = 111 in the TIM1\_SMCR register (see [Figure 45](#)).

**Figure 45. TI2 external clock connection example**



### Procedure

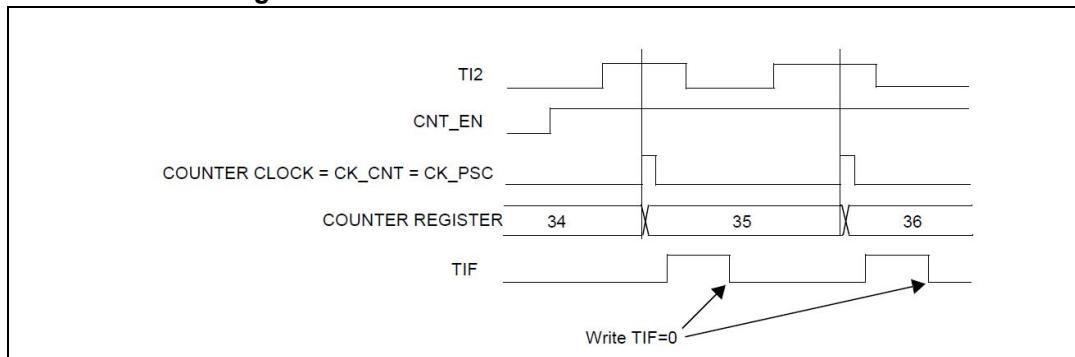
Use the following procedure to configure the up-counter and, for example, to count in response to a rising edge on the TI2 input:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = 01 in the TIM1\_CCMR2 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIM1\_CCMR2 register (if no filter is needed, keep IC2F = 0000).  
*Note: The capture prescaler is not used for triggering, so it does not need to be configured. The CC2S bits do not need to be configured either as they only select the input capture source.*
3. Select rising edge polarity by writing CC2P = 0 in the TIM1\_CCER1 register.
4. Configure the timer in external clock mode 1 by writing SMS = 111 in the TIM1\_SMCR register.
5. Select TI2 as the input source by writing TS = 110 in the TIM1\_SMCR register.
6. Enable the counter by writing CEN = 1 in the TIM1\_CR1 register.

When a rising edge occurs on TI2, the counter counts once and the trigger flag is set (TIF bit in the TIM1\_SR1 register) and an interrupt request can be sent if enabled (depending on the TIE bit in the TIM1\_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 46. Control circuit in external clock mode 1**

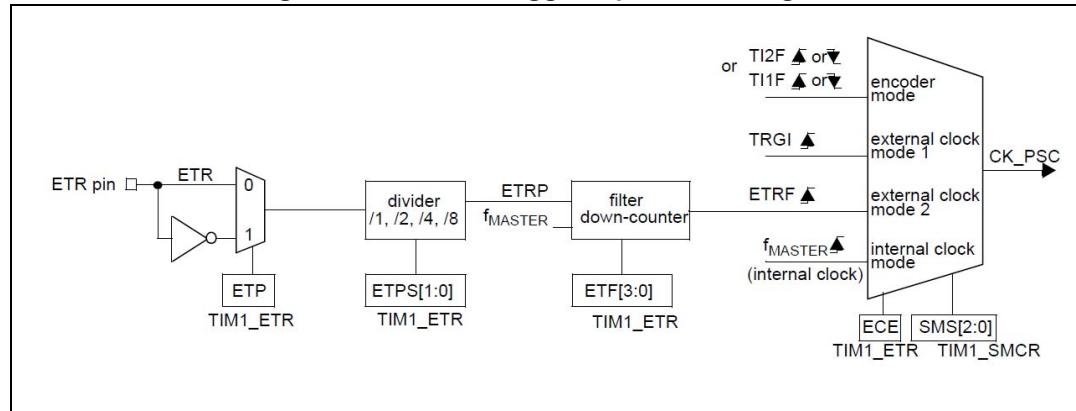


#### 17.4.4 External clock source mode 2

The counter can count at each rising or falling edge on the ETR. This mode is selected by writing ECE = 1 in the TIM1\_ETR register.

The [Figure 47](#) gives an overview of the external trigger input block.

**Figure 47. External trigger input block diagram**



##### Procedure

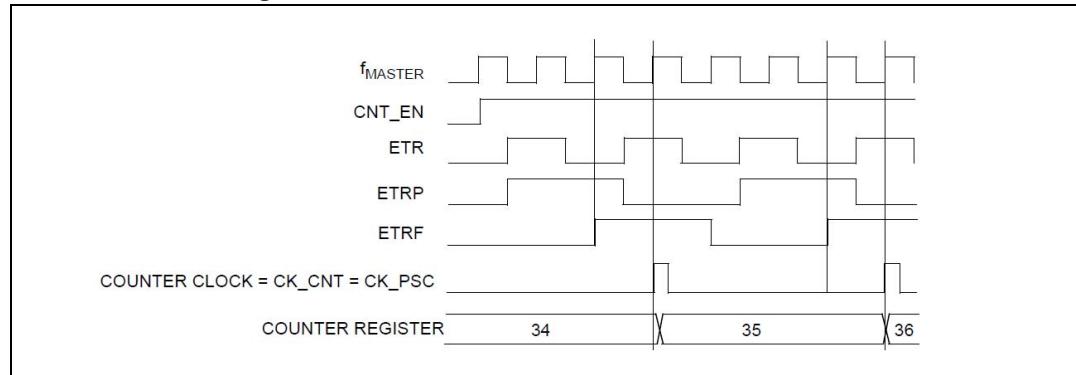
Use the following procedure to configure the up-counter and, for example, to count once every two rising edges on the ETR:

1. As no filter is needed in this example, write ETF[3:0] = 0000 in the TIM1\_ETR register.
2. Set the prescaler by writing ETPS[1:0] = 01 in the TIM1\_ETR register.
3. Select rising edge detection on the ETR pin by writing ETP = 0 in the TIM1\_ETR register.
4. Enable external clock mode 2 by writing ECE = 1 in the TIM1\_ETR register.
5. Enable the counter by writing CEN = 1 in the TIM1\_CR1 register.

The counter counts once every two ETR rising edges.

The delay between the rising edge on the ETR and the actual reset of the counter is due to the resynchronization circuit on the external trigger signal (ETRP).

**Figure 48. Control circuit in external clock mode 2**



### 17.4.5 Trigger synchronization

There are four trigger inputs (refer to [Table 35: Glossary of internal timer signals on page 136](#)):

- ETR
- TI1
- TI2
- TRGO from TIM5/TIM6

The TIM1 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.

##### Procedure

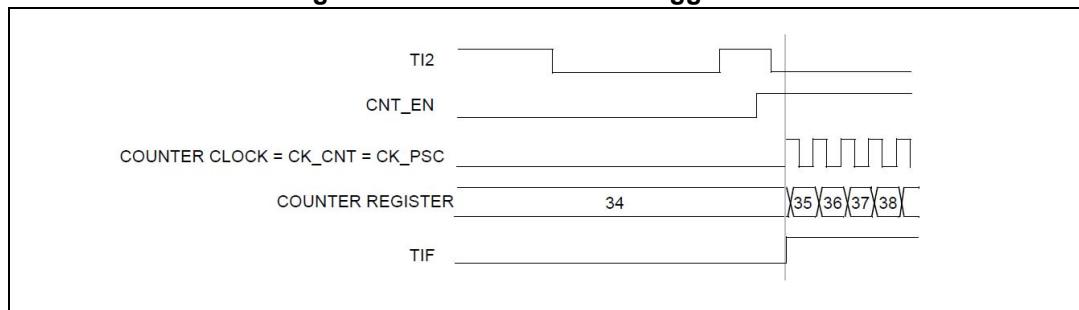
Use the following procedure to start the up-counter in response, for example, to a rising edge on the TI2 input:

1. Configure channel 2 to detect rising edges on TI2. As no filter is required in this example, configure an input filter duration of 0 (IC2F = 0000). The capture prescaler is not used for triggering and does not need to be configured. The CC2S bits select the input capture source and do not need to be configured either. Write CC2P = 0 in the TIM1\_CCER1 register to select rising edge polarity.
2. Configure the timer in trigger mode by writing SMS = 110 in the TIM1\_SMCR register. Select TI2 as the input source by writing TS = 110 in the TIM1\_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 49. 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 TIM1\_CR1 register is low, a UEV is generated. Then all the preloaded registers (TIM1\_ARR, TIM1\_CCR $i$ ) are updated.

#### Example

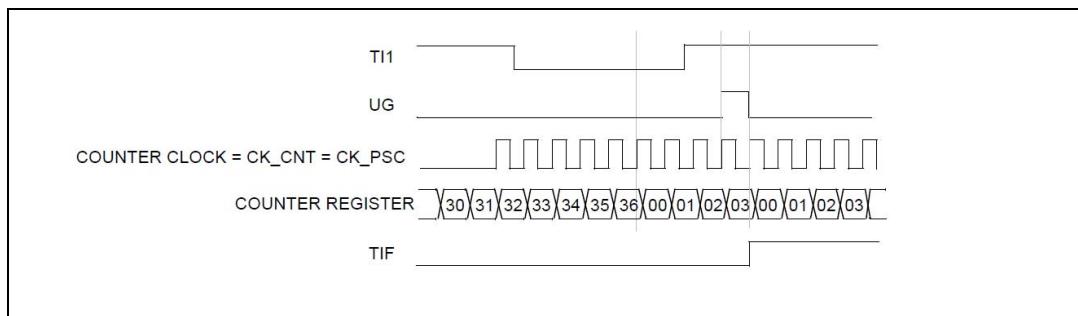
Use the following procedure to clear the up-counter in response to a rising edge on TI1 input:

1. Configure channel 1 to detect rising edges on TI1. As no filter is required in this example, configure an input filter duration of 0 (IC2F = 0000). The capture prescaler is not used for triggering and does not need to be configured. The CC1S bits select the input capture source and do not need to be configured either. Write CC1P = 0 in TIM1\_CCER1 register to validate the polarity (and detect rising edges).
2. Configure the timer in reset mode by writing SMS = 100 in TIM1\_SMCR register. Select TI1 as the input source by writing TS = 101 in the TIM1\_SMCR register.
3. Start the counter by writing CEN = 1 in the TIM1\_CR1 register.

The counter starts counting on the internal clock and behaves normally until the 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 TIM1\_SR1 register) and an interrupt request can be sent if enabled (depending on the TIE in the TIM1\_IER register).

*Figure 50* shows this behavior when the auto-reload register TIM1\_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 50. Control circuit in trigger reset mode**



### Trigger gated mode

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

#### Example

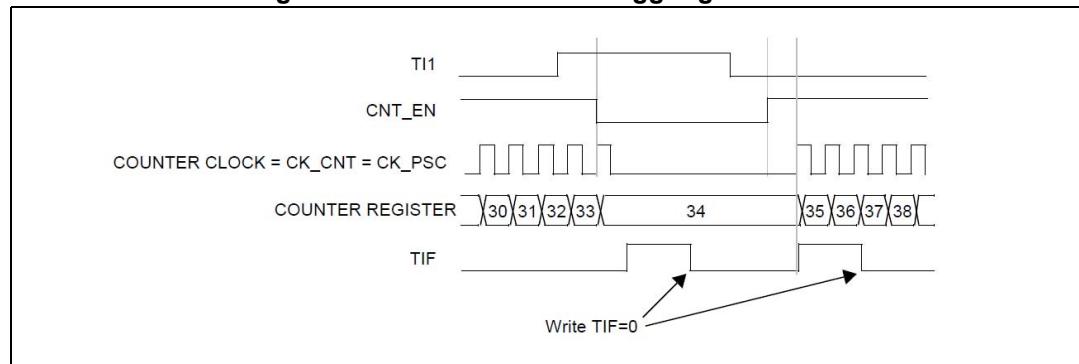
Use the following procedure to enable the up-counter when TI1 input is low:

1. Configure channel 1 to detect low levels on TI1. Configure the input filter duration (IC1F = 0000). The capture prescaler is not used for triggering and does not need to be configured. The CC1S bits select the input capture source and do not need to be configured either. Write CC1P = 1 in the TIM1\_CCER1 register to validate the polarity (and detect low level).
2. Configure the timer in trigger gated mode by writing SMS = 101 in the TIM1\_SMCR register. Select TI1 as the input source by writing TS = 101 in the TIM1\_SMCR register.
3. Enable the counter by writing CEN = 1 in the TIM1\_CR1 register (in trigger gated mode, the counter does not start if CEN = 0 irrespective of the trigger input level).

The counter starts counting on the internal clock as long as TI1 is low. It stops as soon as TI1 becomes high. The TIF flag is set 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 51. Control circuit in trigger gated mode**



### Combining trigger modes with external clock mode 2

External clock mode 2 can be used with another trigger mode. For example, the ETR can be used as the external clock input, and a different input can be selected as trigger input (in trigger standard mode, trigger reset mode, or trigger gated mode). Care must be taken not to select ETR as TRGI (through the TS bits in the TIM1\_SMCR register).

#### Example

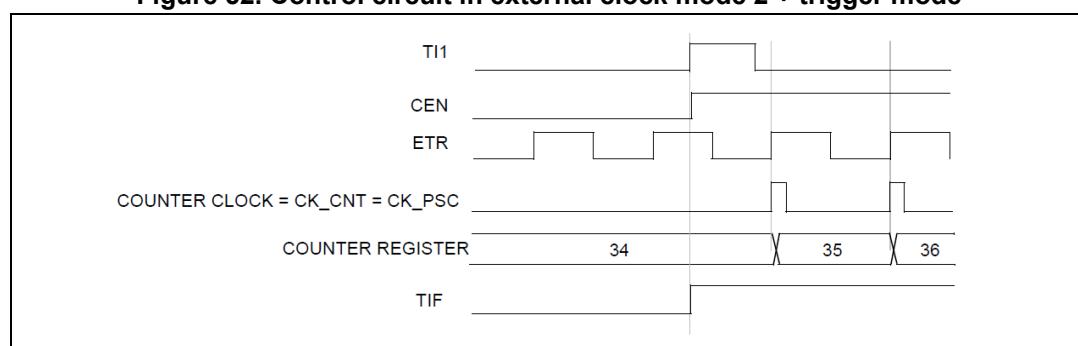
Use the following procedure to enable the up-counter at each rising edge on the ETR as soon as a rising edge occurs on TI1 (standard trigger mode with external ETR clock):

1. Configure the external trigger input circuit by writing to the TIM1\_ETR register. Write ETF = 0000 (no filter needed in this example). Write ETPS = 00 to disable the prescaler, write ETP = 0 to detect rising edges on the ETR, and write ECE = 1 to enable external clock mode 2.
2. Configure channel 1 to detect rising edges on TI1. Configure the input filter duration (IC1F = 0000). The capture prescaler is not used for triggering and does not need to be configured. The CC1S bits select the input capture source and do not need to be configured either. Write CC1P = 0 in the TIM1\_CCER1 register to select rising edge polarity.
3. Configure the timer in trigger mode by writing SMS = 110 in the TIM1\_SMCR register. Select TI1 as the input source by writing TS = 101 in the TIM1SMCR register.

A rising edge on TI1 enables the counter and sets the TIF flag. Consequently, the counter counts on the 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 the ETR and the actual reset of the counter is due to the resynchronization circuit on the ETRP signal.

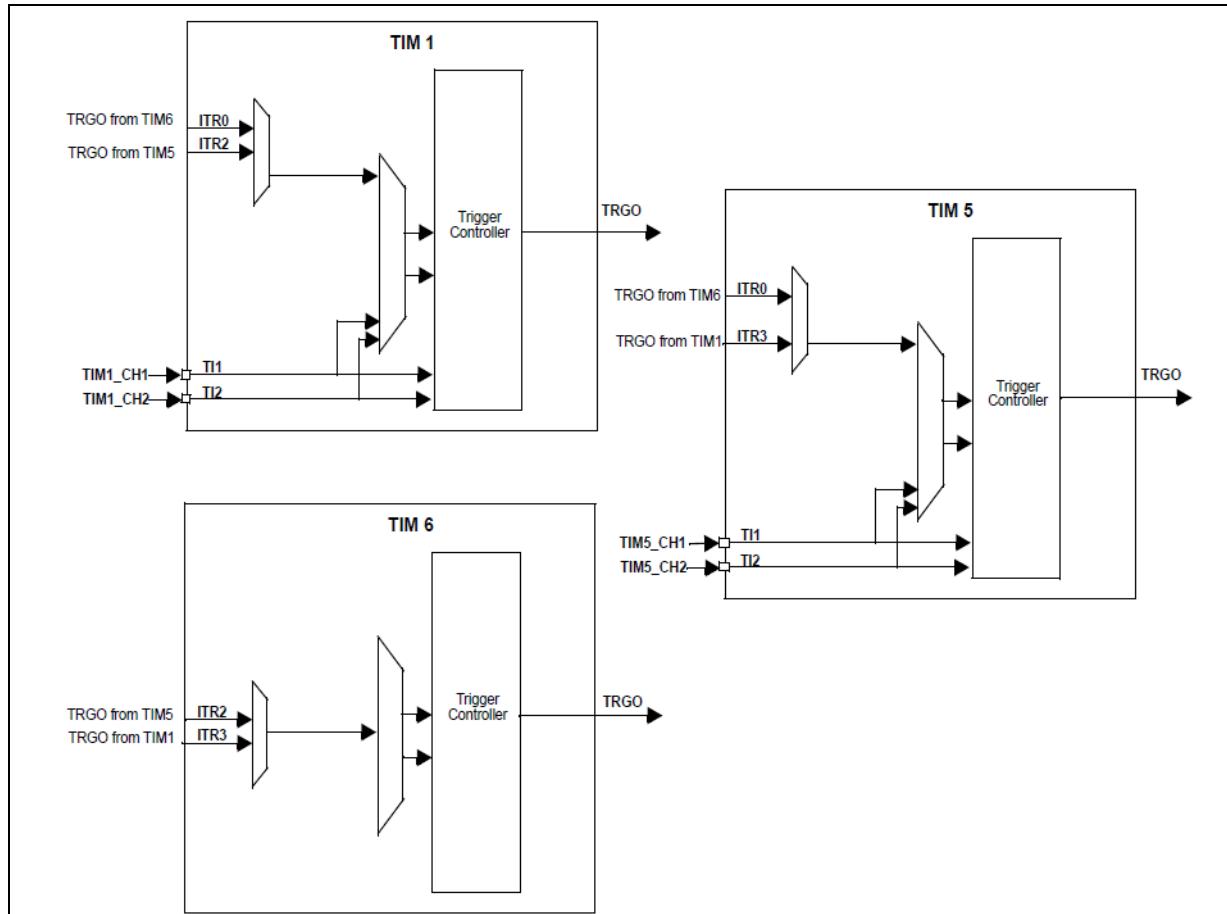
**Figure 52. Control circuit in external clock mode 2 + trigger mode**



#### 17.4.6 Synchronization between TIM1, TIM5 and TIM6 timers

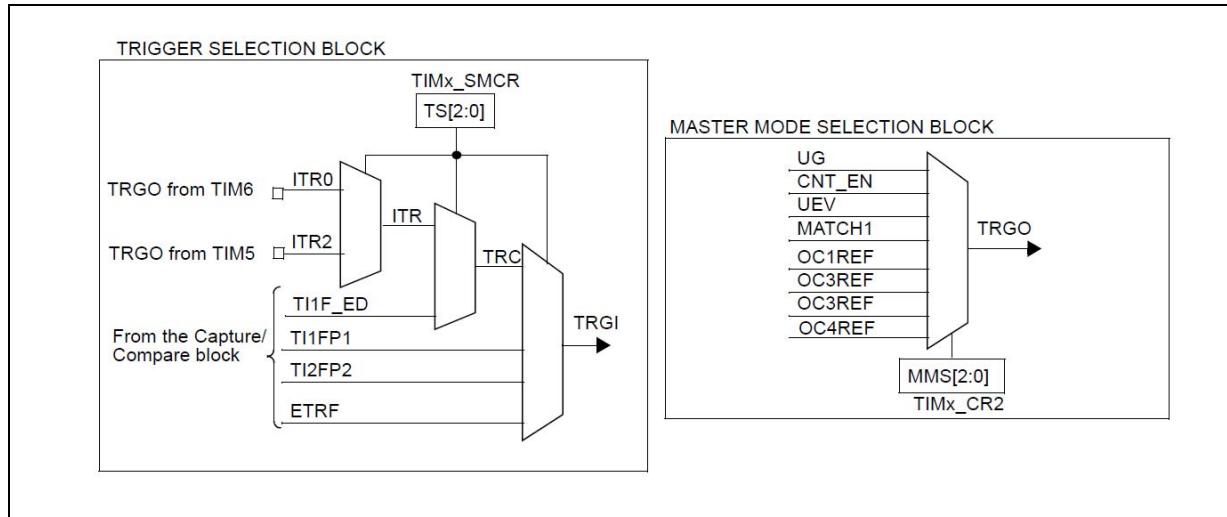
On STM8AF and STM8S low-density devices, the timers are linked together internally for timer synchronization or chaining. When one timer is configured in master mode, it can output a trigger (TRGO) to reset, start, stop, or clock the counter of any other timer configured in slave mode.

Figure 53. Timer chaining system implementation example



*Figure 54* presents an overview of the trigger selection and the master mode selection blocks.

**Figure 54. Trigger/master mode selection blocks**



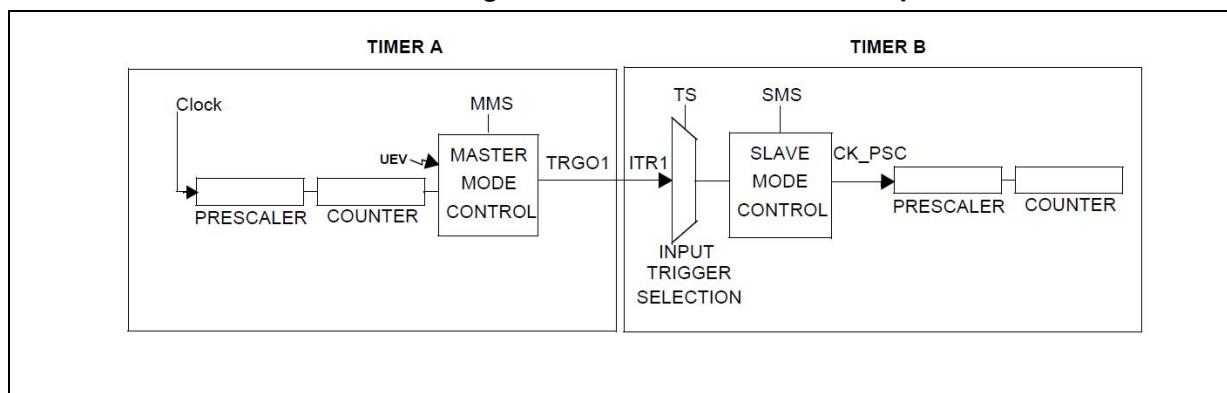
### Using one timer as prescaler for another timer

Refer to *Figure 55* to see how timer A can be configured to act as a prescaler for timer B.

1. Configure timer A in master mode so that it outputs a periodic trigger signal on each 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 the internal trigger. Select this through the TS bits in the **TIMx\_SMCR** register (see TS[2:0] bit definitions in **TIMx\_SMCR** register).
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. 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.

**Figure 55. Master/slave timer example**



## Using one timer to enable another timer

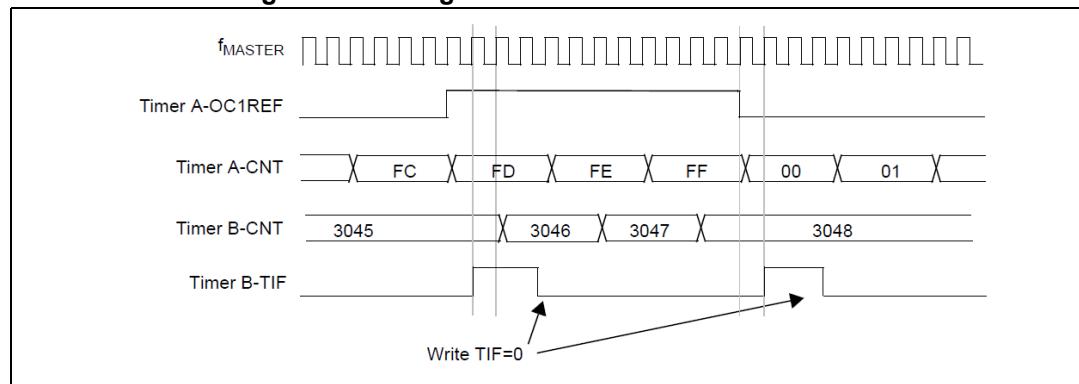
### Example 1

The enable of timer B is controlled with the output compare 1 of timer A (refer to [Figure 56](#) 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 four by the prescaler compared to  $f_{\text{MASTER}}$  ( $f_{\text{CK\_CNT}} = f_{\text{MASTER}}/4$ ).

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 (see TS[2:0] bit definitions in 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 56. Gating timer B with OC1REF of timer A**



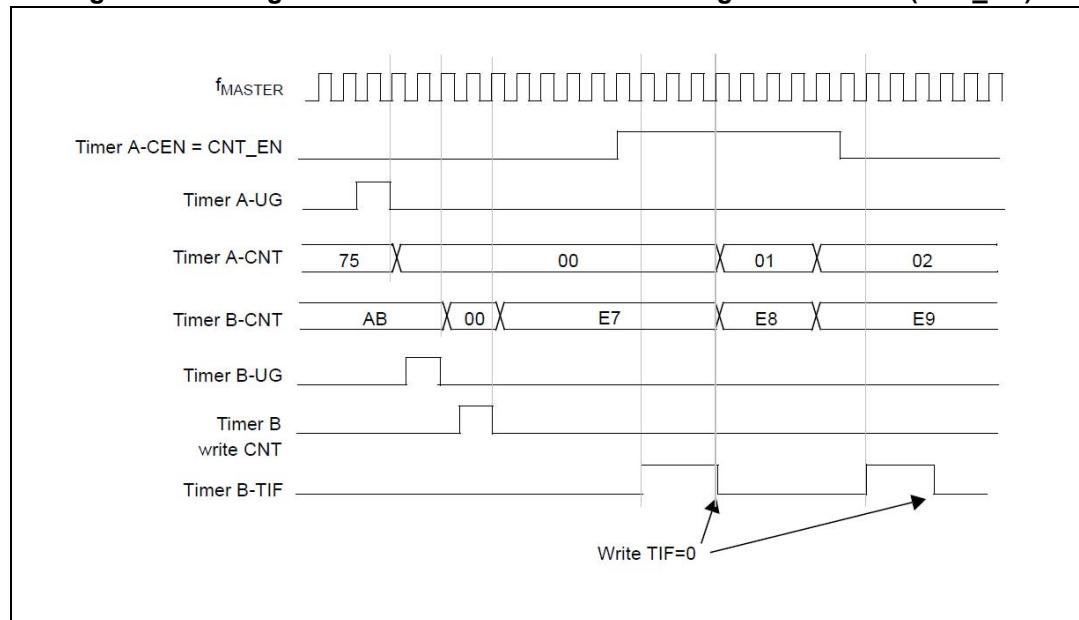
In [Figure 56](#), the timer B counter and prescaler are not initialized before being started. Therefore, they start counting from their current value. It is possible to start from a given value by resetting both timers before starting timer A. In this case, any value can be written in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx\_EGR registers.

### Example 2

Timer A and timer B are synchronized. 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 (see TS[2:0] bit definitions in 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 57. Gating timer B with the counter enable signal of timer A (CNT\_EN)**



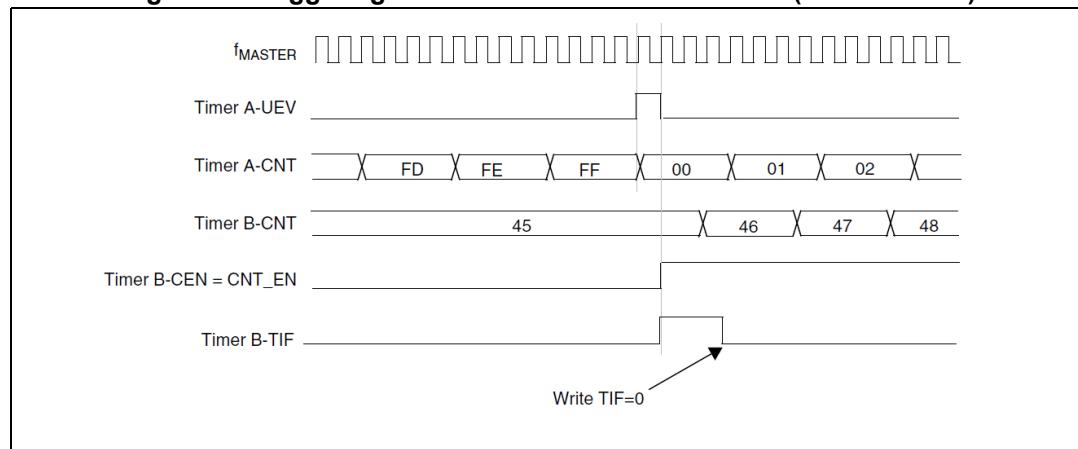
### Using one timer to start another timer

#### Example 1

The enable of timer B is set with the UEV of timer A (refer to [Figure 55](#) for connections). Timer B starts counting from its current value (which can be non-zero) on the divided internal clock as soon as the UEV is generated by timer A. When timer B receives the trigger signal, its CEN bit is automatically set and the counter counts until 0 is written to the CEN bit in the TIM1\_CR1 register. Both counter clock frequencies are divided by four by the prescaler compared to  $f_{MASTER}$  ( $f_{CK\_CNT} = f_{MASTER}/4$ ).

1. Configure timer A master mode to send its UEV as trigger output (MMS = 010 in the TIM1\_CR2 register).
2. Configure the timer A period (TIM1\_ARR registers)
3. Configure timer B to get the input trigger from timer A (see TS[2:0] bit definitions in TIM1\_SMCR register).
4. Configure timer B in trigger mode (SMS = 110 in TIM1\_SMCR register)
5. Start timer A by writing 1 in the CEN bit (TIM1\_CR1 register)

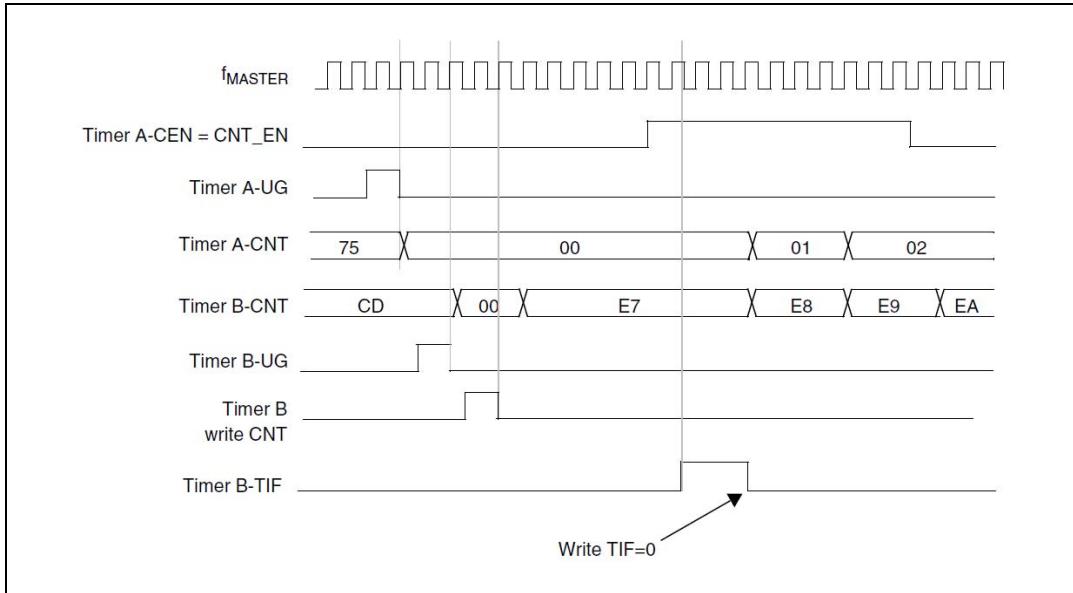
**Figure 58. Triggering timer B with the UEV of timer A (TIMERA-UEV)**



### Example 2

As in the previous example, both counters can be initialized before starting to count. [Figure 59](#) shows the behavior, with the same configuration as in [Figure 57](#), but, in trigger standard mode instead of trigger gated mode (SMS = 110 in the TIM1\_SMCR register).

**Figure 59. Triggering timer B with counter enable CNT\_EN of timer A**



### Starting 2 timers synchronously in response to an external trigger

#### Example

The enable of timer A is set when its TI1 input rises and the enable of timer B is set with the enable of timer A (refer to [Figure 55](#) 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).

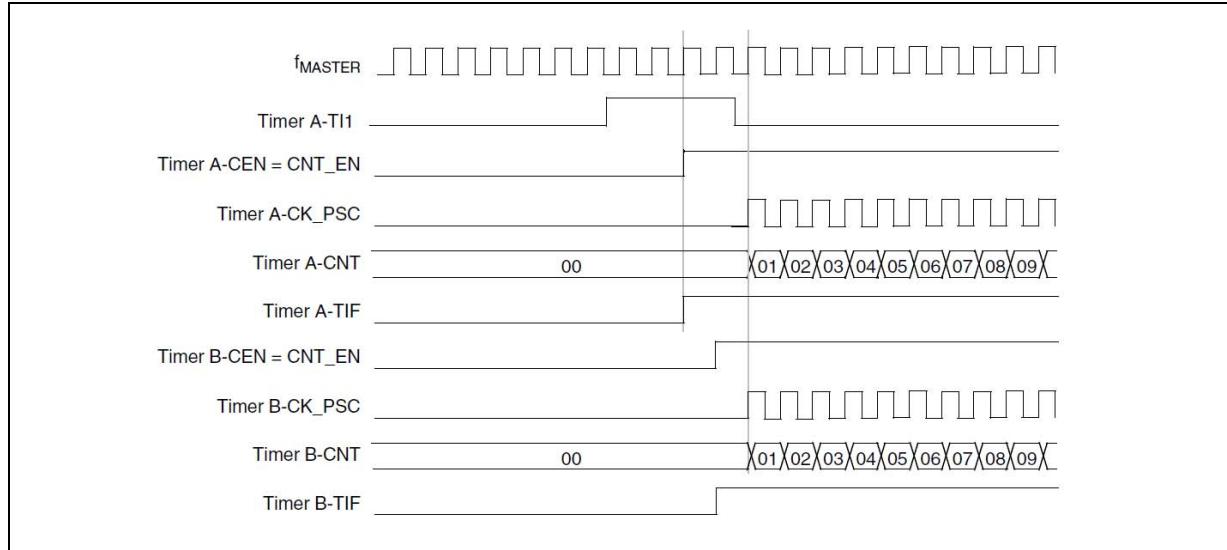
1. Configure timer A master mode to send its enable as trigger output (MMS = 001 in the TIMx\_CR2 register).
2. Configure timer A slave mode to get the input trigger from TI1 (TS = 100 in the TIMx\_SMCR register).
3. Configure timer A in trigger mode (SMS = 110 in the TIMx\_SMCR register)
4. Configure 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 (see TS[2:0] bit definitions in 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 start 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 start from 0, but an offset can easily be inserted between them by writing to any of the counter registers (TIMx\_CNT). It can be seen that the master/slave mode inserts a delay between CNT\_EN and CK\_PSC on timer A.*

Figure 60. Triggering Timer A and B with Timer A TI1 input

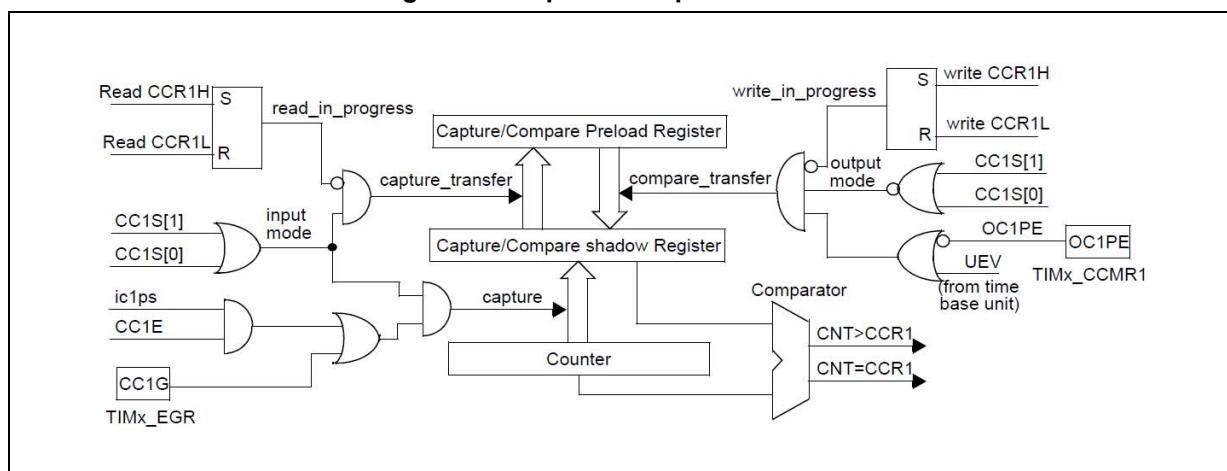


## 17.5 TIM1 capture/compare channels

The timer I/O pins (TIM1\_CC<sub>i</sub>) can be configured either for input capture or output compare functions. The choice is made by configuring the CC<sub>i</sub>S channel selection bits in the capture/compare channel mode registers (TIM1\_CCMR<sub>i</sub>), 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 61. Capture/compare channel 1 main circuit

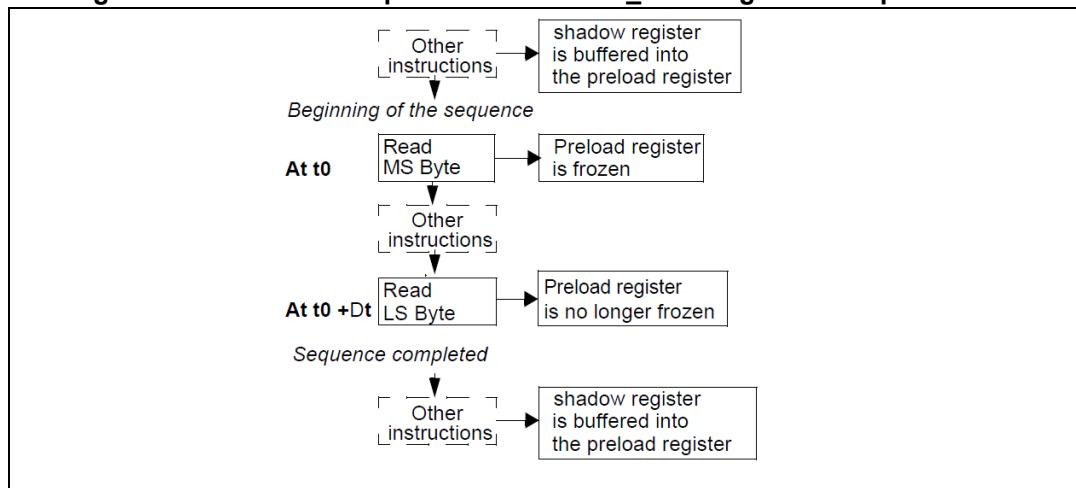


The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register. In capture mode, captures are made in the shadow register, which is copied into the preload register. In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

When the channel is configured in output mode ( $CCiS = 00$  in the  $TIM1\_CCMRi$  registers), the  $TIM1\_CCRi$  registers can be accessed without any restriction.

When the channel is configured in input mode, the sequence for reading the  $TIM1\_CCRi$  registers is the same as for the counter (see [Figure 62](#)). When a capture occurs, the content of the counter is captured into the  $TIM1\_CCRi$  shadow registers. Then this value is loaded into the preload register, except during a read sequence, when the preload register is frozen.

**Figure 62. 16-bit read sequence for the  $TIM1\_CCRi$  register in capture mode**



[Figure 62](#) shows the sequence for reading the  $CCRi$  registers in the 16-bit timers. This buffered value remains unchanged until the 16-bit read sequence is completed.

After a complete read sequence, if only the  $TIM1\_CCRiL$  registers are read, they return 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.

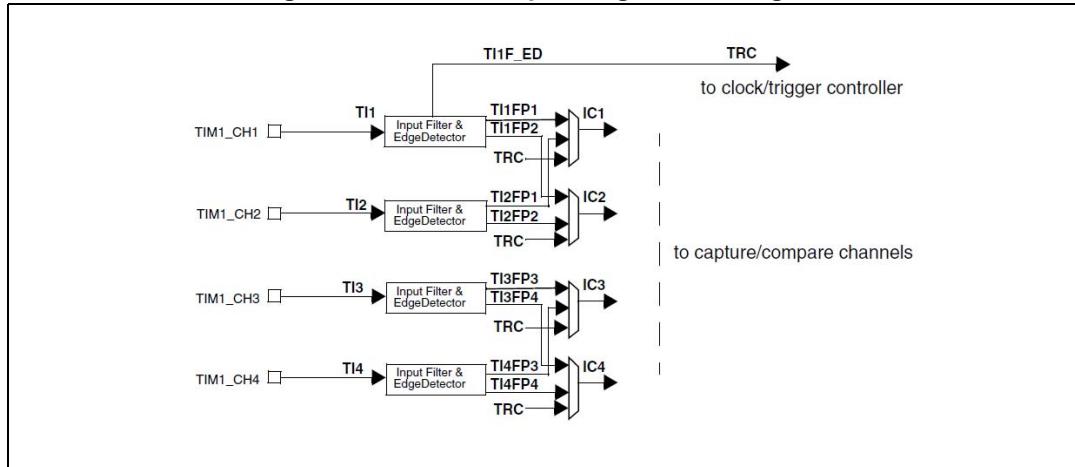
### 17.5.1 Write sequence for 16-bit $TIM1\_CCRi$ registers

16-bit values are loaded in the  $TIM1\_CCRi$  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 is written. Do not use the LDW instruction, as this writes the LS byte first, and produces incorrect results in this case.

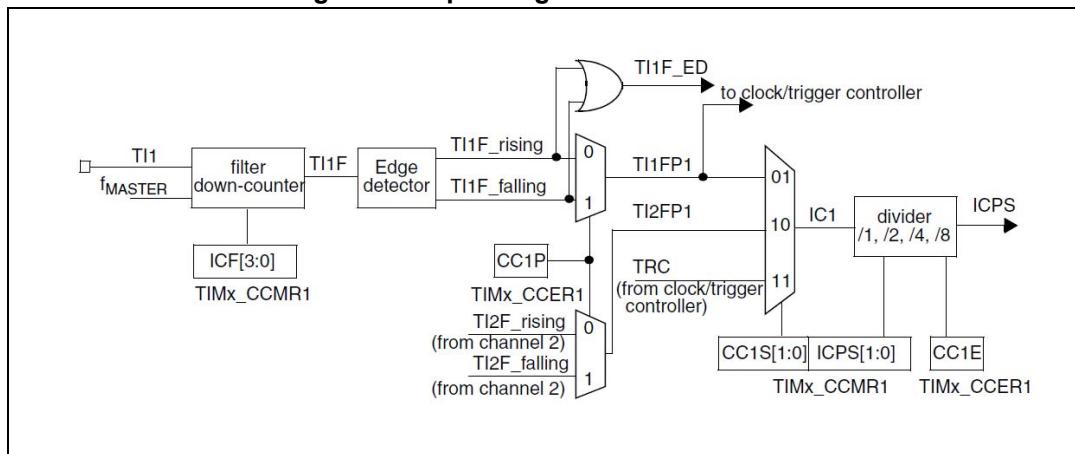
### 17.5.2 Input stage

**Figure 63. Channel input stage block diagram**



*Figure 64* shows how the input stage samples the corresponding  $TI_i$  input to generate a filtered signal  $TI_iF$ . Then, an edge detector with polarity selection, generates a signal ( $TI_iFPn$ ) which can be used as trigger input by the clock/trigger controller or as the capture command. The signal is prescaled before entering the capture register ( $IC_iPS$ ).

**Figure 64. Input stage of TIM 1 channel 1**



### 17.5.3 Input capture mode

In input capture mode, the capture/compare registers (TIM1\_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 (TIM1\_SR1 register) is set.

An interrupt can be sent if it is enabled, by setting the CC $i$ E bits in the TIM1\_IER register. If a capture occurs while the CC $i$ IF flag is already high, the overcapture flag CC $i$ OF (TIM1\_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 registers. CC $i$ OF is cleared by writing it to 0.

#### Procedure

The following procedure shows how to capture the counter value in TIM1\_CCR1, for example, when TI1 input rises.

1. Select the active input: For example, to link the TIM1\_CCR1 register to the TI1 input, write the CC1S bits to 01 in the TIM1\_CCMR1 register. This configures the channel in input mode and the TIM1\_CCR1 register becomes read-only.
2. Program the required input filter duration for the signal to be connected to the timer. This is done for each TI $i$  input using the IC $i$ F bits in the TIM1\_CCMR $i$  registers. For example, if the input signal is unstable for up to five t<sub>MASTER</sub> cycles when it toggles, the filter duration must be performed for longer than five clock cycles. The filter bits allow a duration of eight cycles to be selected by writing them to 0011 in the TIMx\_CCMR1 register. With this filter setting, a transition on TI1 is valid only when eight consecutive samples with the new level have been detected (sampled at f<sub>MASTER</sub> frequency).
3. Select the edge of the active transition on the TI1 channel by writing the CC1P bit to 0 in the TIM1\_CCER1 register (rising edge in this case).
4. Program the input prescaler. In this example, the capture needs to be performed at each valid transition, so the prescaler is disabled (write the IC1PS bits to 00 in the TIM1\_CCMR1 register).
5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIM1\_CCER1 register.
6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIM1\_IER register.

When an input capture occurs:

- The TIM1\_CCR1 register gets the value of the counter on the active transition
- The input capture flag (CC1IF) is set. The overcapture flag (CC1OF) is also set if at least two consecutive captures occur while the flag remains uncleared.
- 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 avoids missing an overcapture which could occur after reading the flag and before reading the data.

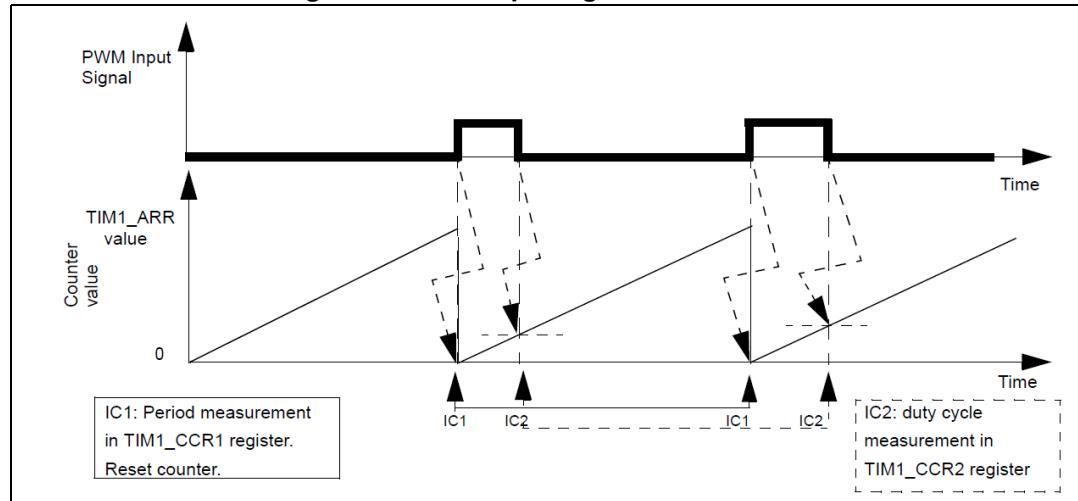
*Note:* IC interrupts can be generated by software by setting the corresponding CC $i$ G bits in the TIM1\_EGR register.

## PWM input signal measurement

This mode is a particular case of input capture mode (see [Figure 65](#)). The procedure is the same except:

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

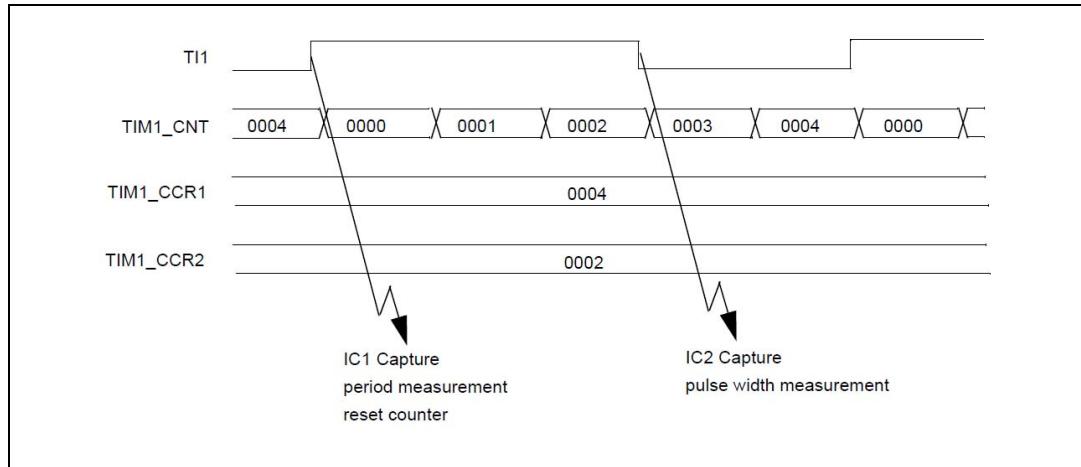
**Figure 65. PWM input signal measurement**



### Procedure

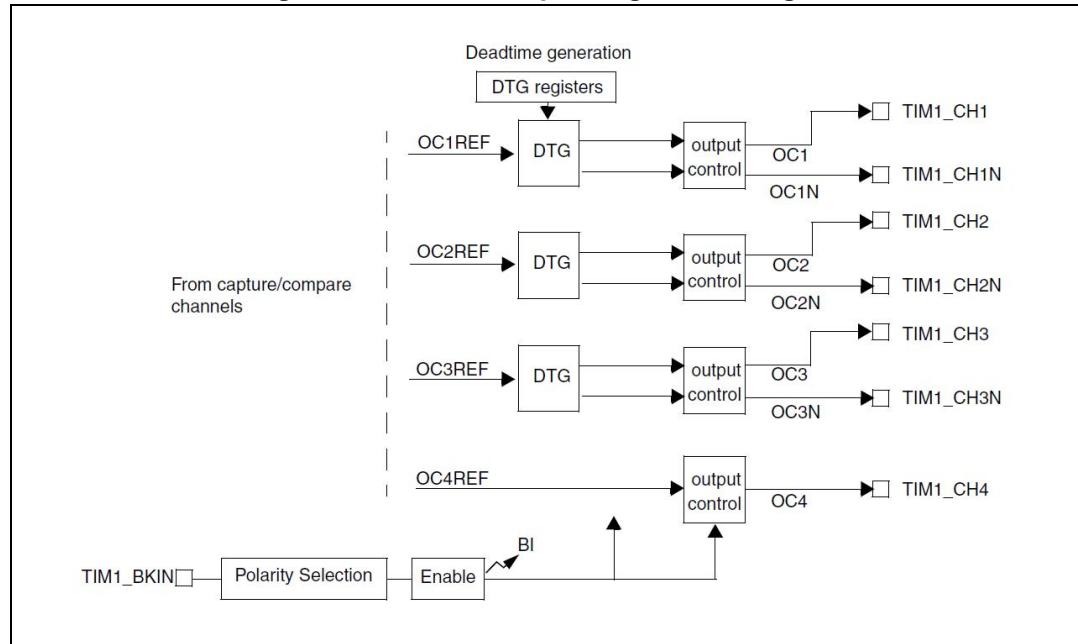
Depending on the f<sub>MASTER</sub> frequency and prescaler value, the period (in the TIM1\_CCR1 register) can be measured and the duty cycle (in the TIM1\_CCR2 register) of the PWM can be applied on TI1 using the following procedure:

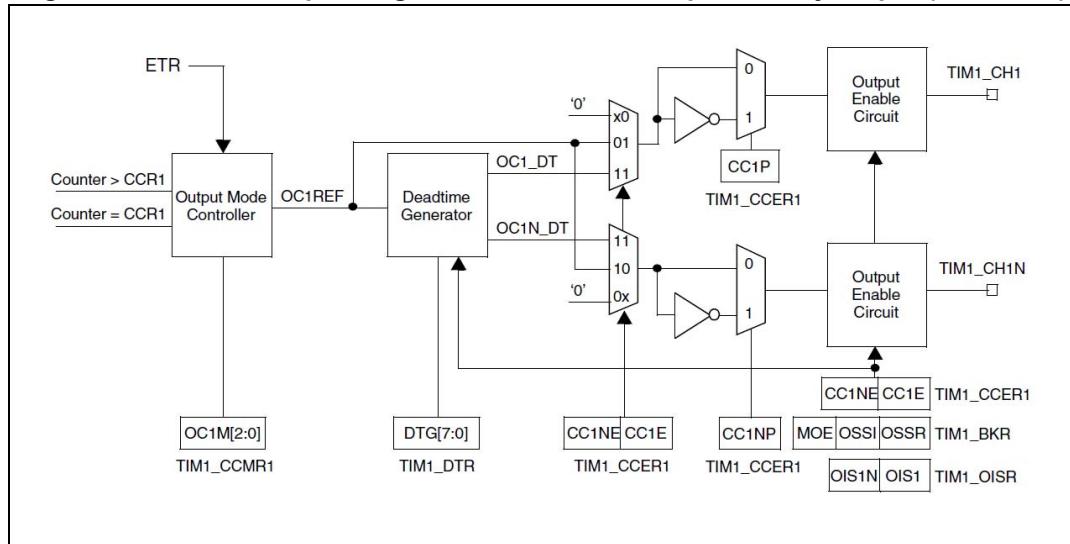
1. Select the active input capture or trigger input for TIM1\_CCR1 by writing the CC1S bits to 01 in the TIM1\_CCMR1 register (TI1FP1 selected).
2. Select the active polarity for TI1FP1 (used for both capture and counter clear in TIMx\_CCR1) by writing the CC1P bit to 0 (TI1FP1 active on rising edge).
3. Select the active input for TIM1\_CCR2 by writing the CC2S bits to 10 in the TIM1\_CCMR2 register (TI1FP2 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIM1\_CCR2) by writing the CC2P bit to 1 (TI1FP2 active on falling edge).
5. Select the valid trigger input by writing the TS bits to 101 in the TIM1\_SMCR register (TI1FP1 selected).
6. Configure the clock/trigger controller in reset mode by writing the SMS bits to 100 in the TIM1\_SMCR register.
7. Enable the captures by writing the CC1E and CC2E bits to 1 in the TIM1\_CCER1 register.

**Figure 66. 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 67. Channel output stage block diagram**

**Figure 68. Detailed output stage of channel with complementary output (channel 1)**

### 17.5.5 Forced output mode

In output mode (CC/S bits = 00 in the TIM1\_CCMR*i* registers), 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, write 101 in the OC/M bits in the corresponding TIM1\_CCMR*i* registers. OC/REF is forced high (OC/REF is always active high) and the OC*i* output is forced high or low depending on the CC/P polarity bits.

For example, if CC/P = 0 (OC*i* active high) => OC*i* is forced high.

The OC/REF signal can be forced low by writing the OC/M bits to 100 in the TIMx\_CCMRx registers.

Nevertheless, the comparison between the TIM1\_CCR*i* shadow registers 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/M = 000),
  - Is set active (OC/M = 001),
  - Is set inactive (OC/M = 010)
  - Toggles (OC/M = 011)
- A flag is set in the interrupt status register (CC*i*F bits in the TIM1\_SR1 register).
- An interrupt is generated if the corresponding interrupt mask is set (CC*i*E bits in the TIM1\_IER register).

The output compare mode is defined by the OC*i*M bits in the TIM1\_CCMR*i* registers. The active or inactive level polarity is defined by the CC*i*P bits in the TIM1\_CCER*i* registers.

The TIM1\_CCR*i* registers can be programmed with or without preload registers using the OC<sub>i</sub>PE bits in the TIM1\_CCMR*i* registers.

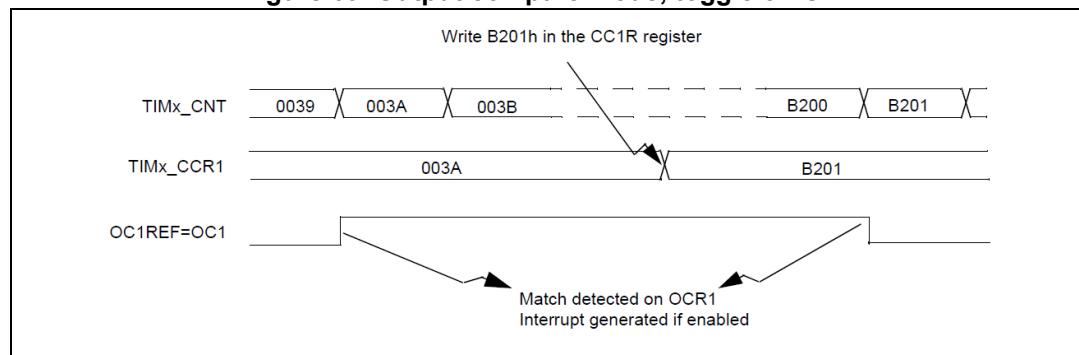
In output compare mode, the UEV has no effect on the OC*i*REF and OC*i* output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse.

#### Procedure

1. Select the counter clock (internal, external, or prescaler).
2. Write the desired data in the TIM1\_ARR and TIM1\_CCR*i* registers.
3. Set the CC*i*E bits if an interrupt request is to be generated.
4. Select the output mode as follows:
  - Write OC*i*M = 011 to toggle the OC*i* output pin when CNT matches CCR*i*
  - Write OC<sub>i</sub>PE = 0 to disable the 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 TIM1\_CCR*i* registers can be updated at any time by software to control the output waveform, provided that the preload registers are not enabled (OC<sub>i</sub>PE = 0). Otherwise, the TIMx\_CCR*i* shadow registers are updated only at the next UEV (see example in [Figure 69](#)).

**Figure 69. 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 TIM1\_ARR register and a duty cycle determined by the value of the TIM1\_CCR*i* registers.

The PWM mode can be selected independently on each channel (one PWM per OC*i* output) by writing 110 (PWM mode 1) or 111 (PWM mode 2) in the OC*M* bits in the TIM1\_CCMR*i* registers. The corresponding preload register must be enabled by setting the OC*i*PE bits in the TIM1\_CCMR*i* registers. The auto-reload preload register (in up-counting or center-aligned modes) may be optionally enabled by setting the ARPE bit in the TIM1\_CR1 register.

As the preload registers are transferred to the shadow registers only when an UEV occurs, all registers have to be initialized by setting the UG bit in the TIM1\_EGR register before starting the counter.

OC*i* polarity is software programmable using the CC*i*P bits in the TIM1\_CCER*i* registers. It can be programmed as active high or active low. The OC*i* output is enabled by a combination of CC*i*E, MOE, OIS*i*, OSSR and OSSI bits (TIM1\_CCER*i* and TIM1\_BKR registers). Refer to the TIM1\_CCER*i* register descriptions for more details.

In PWM mode (1 or 2), TIM1\_CNT and TIM1\_CCR*i* are always compared to determine whether TIM1\_CCR*i* ≤ TIM1\_CNT or TIM1\_CNT ≤ TIM1\_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 TIM1\_CR1 register.

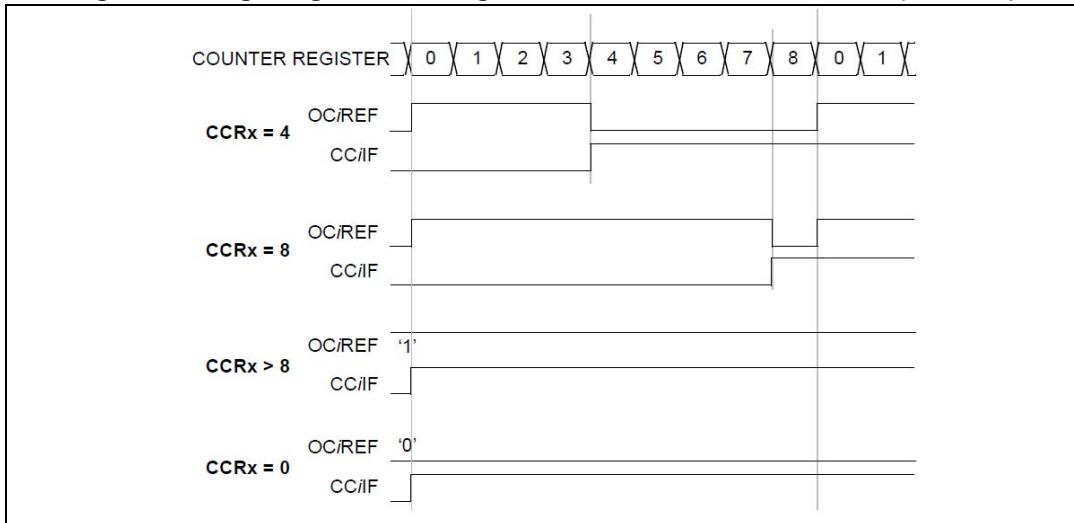
#### PWM edge-aligned mode

Up-counting configuration

Up-counting is active when the DIR bit in the TIM1\_CR1 register is low.

##### Example

This example uses PWM mode 1. The reference PWM signal, OC*i*REF, is high as long as TIM1\_CNT < TIM1\_CCR*i*. Otherwise, it becomes low. If the compare value in TIM1\_CCR*i* is greater than the auto-reload value (in TIM1\_ARR) then OC*i*REF is held at 1. If the compare value is 0, OC*i*REF is held at 0. [Figure 70](#) shows some edge-aligned PWM waveforms in an example where TIM1\_ARR = 8.

**Figure 70. Edge-aligned counting mode PWM mode 1 waveforms (ARR = 8)**

Down-counting configuration

Down-counting is active when the DIR bit in the TIM1\_CR1 register is high. Refer to [Down-counting mode on page 145](#)

In PWM mode 1, the reference signal OC/REF is low as long as  $\text{TIM1\_CNT} > \text{TIM1\_CCR}_i$ . Otherwise, it becomes high. If the compare value in the TIM1\_CCR<sub>i</sub> registers is greater than the auto-reload value in the TIM1\_ARR register, OC/REF is held at 1. Zero percent PWM is not possible in this mode.

### PWM center-aligned mode

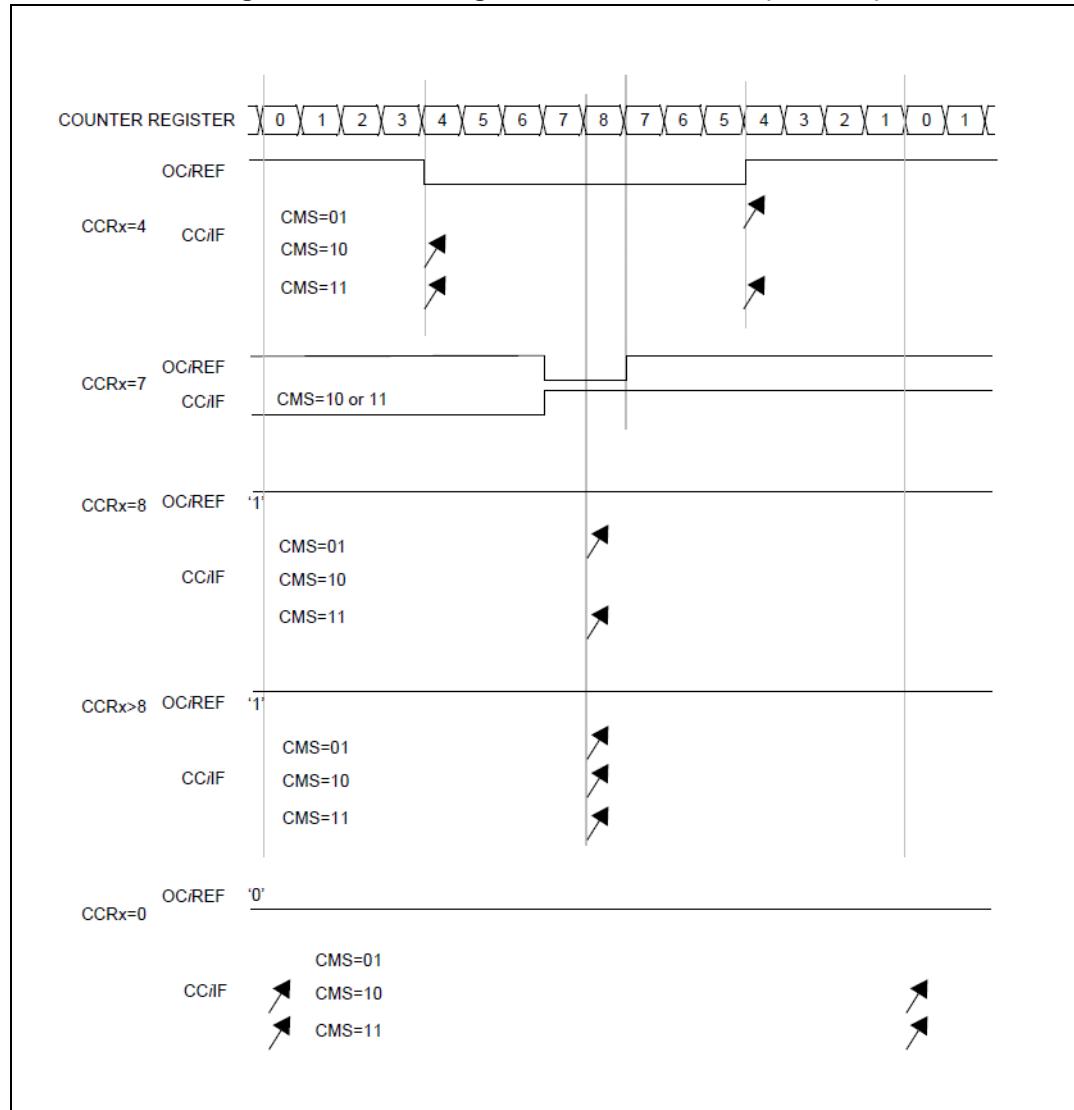
Center-aligned mode is active when the CMS bits in the TIM1\_CR1 register are different from 00 (all the remaining configurations have the same effect on the OC/REF/OC<sub>i</sub> signals).

The compare flag is set when the counter counts up, down, or up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIM1\_CR1 register is updated by hardware and is read-only in this mode (refer to [Center-aligned mode \(up/down counting\) on page 147](#)).

*Figure 71* shows some center-aligned PWM waveforms in an example where:

- TIM1\_ARR = 8,
- PWM mode is PWM mode 1
- The flag is set (arrow symbol in *Figure 71*) in three different cases:
  - When the counter counts down (CMS = 01)
  - When the counter counts up (CMS = 10)
  - When the counter counts up and down (CMS = 11)

Figure 71. Center-aligned PWM waveforms (ARR = 8)



### One-pulse mode

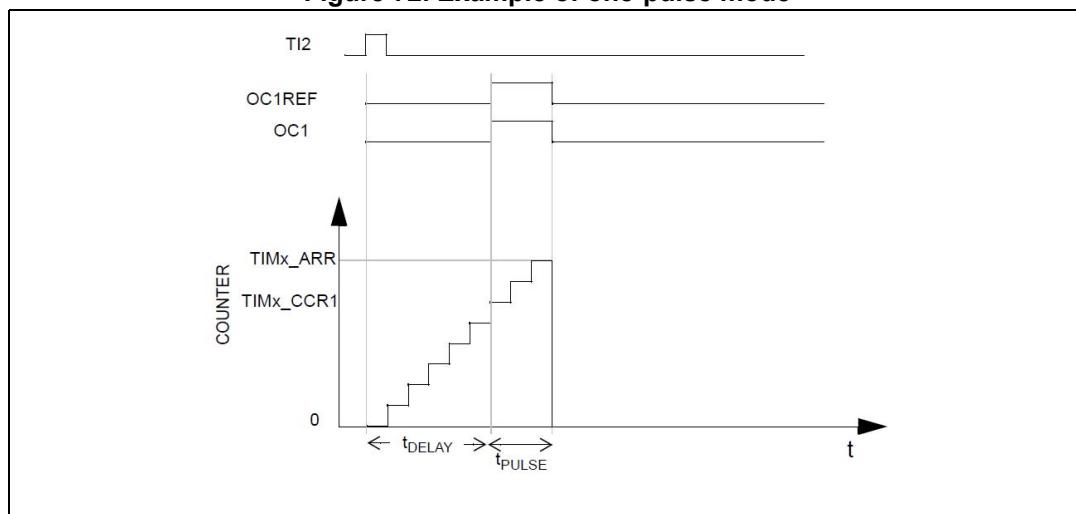
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the clock/trigger controller. Generating the waveform can be done in output compare mode or PWM mode. Select one-pulse mode by setting the OPM bit in the TIM1\_CR1 register. This makes the counter stop automatically at the next 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:  $CNT < CCR_i \leq ARR$  (in particular,  $0 < CCR_i$ ),
- In down-counting:  $CNT > CCR_i$

**Figure 72. Example of one-pulse mode**



#### Example

This example shows how 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.

Follow the procedure below to use IC2 as trigger 1:

- Map IC2 on TI2 by writing CC2S = 01 in the TIM1\_CCMR2 register
- IC2 must detect a rising edge, so write CC2P = 0 in the TIM1\_CCER1 register
- Configure IC2 as trigger for the clock/trigger controller (TRGI) by writing TS = 110 in the TIM1\_SMCR register.
- IC2 is used to start the counter by writing SMS to 110 in the TIM1\_SMCR register (trigger mode).

The OPM waveform is defined by writing the compare registers (taking into account the clock frequency and the counter prescaler) as follows:

- The  $t_{DELAY}$  is defined by the value written in the TIM1\_CCR1 register
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value (TIM1\_ARR - TIM1\_CCR1).
- 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, enable PWM mode 2 by writing OCiM = 111 in the TIM1\_CCMR1 register. Alternatively, enable the preload registers by writing OC1PE = 1 in the TIM1\_CCMR1 register and ARPE = 0 in the TIM1\_CR1 register (optional). In this case, write the compare value in the TIM1\_CCR1 register and write the auto-reload value in the TIM1\_ARR register. Then, generate an update by setting the UG bit and wait for an external trigger event on TI2. CC1P is written to 0 in this example.

In the example outlined above, the DIR and CMS bits in the TIM1\_CR1 register should be low.

As only one pulse is required, write 1 in the OPM bit in the TIM1\_CR1 register to stop the counter at the next UEV (when the counter rolls over from the auto-reload value back to 0).

Particular case: OCi fast enable

In one-pulse mode, the edge detection on the TI*i* input sets the CEN bit which enables the counter. Then, a comparison between the counter and the compare value makes the output toggle. However, several clock cycles are needed for these operations and this affects the minimum delay ( $t_{DELAY}$  min) that can be obtained.

To output a waveform with the minimum delay, set the OC/FE bits in the TIM1\_CCMR*i* registers. OC*i*REF (and OC*i*) are forced in response to the stimulus, without taking the comparison into account. The new level of OC*i*REF (and OC*i*) is the same as if a compare match had occurred. The OC/FE bits acts only if the channel is configured in PWM1 or PWM2 mode.

### Complementary outputs and deadtime insertion

TIM1 can output two complementary signals per channel. It also manages the switching-off and switching-on instants of the outputs (see [Figure 31: TIM1 general block diagram on page 140](#)).

This time is generally known as deadtime. Deadtimes must be adjusted depending on the characteristics of the devices connected to the outputs (example, intrinsic delays of level-shifters, delays due to power switches).

The polarity of the outputs can be selected (main output OC*i* or complementary OC*i* N) independently for each output. This is done by writing to the CC*i* P and CC*i* NP bits in the TIM1\_CCER*i* registers.

The complementary signals OC*i* and OC*i* N are activated by a combination of several control bits: The CC*i* E and CC*i* NE bits in the TIM1\_CCER*i* register and, if the break feature is implemented, the MOE, OIS*i*, OIS*i* N, OSS*i*, and OSSR bits in the TIM1\_BKR register. Refer to [Table 38: Output control for complementary OC\*i\* and OC\*i\* N channels with break feature on page 207](#) for more details. In particular, the deadtime is activated when switching to the IDLE state (when MOE falls to 0).

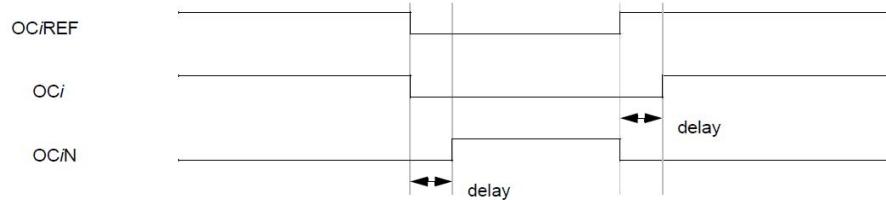
Deadtime insertion is enabled by setting the CC*i* E and CC*i* NE bits, and the MOE bit if the break circuit is present. Each channel embeds an 8-bit deadtime generator. It generates two outputs: OC*i* and OC*i* N from a reference waveform, OC*i* REF. If OC*i* and OC*i* N are active high:

- The OC*i* output signal is the same as the reference signal except for the rising edge, which is delayed relative to the reference rising edge.
- The OC*i* N output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference falling edge.

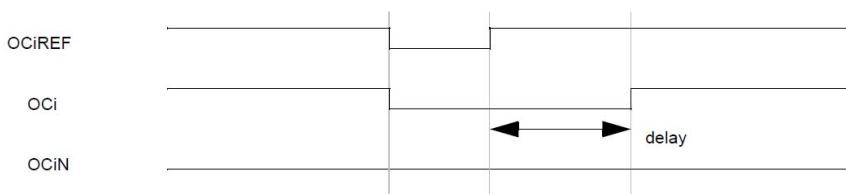
If the delay is greater than the width of the active output (OC*i* or OC*i* N), the corresponding pulse is not generated.

*Figure 73*, *Figure 74*, and *Figure 75* show the relationships between the output signals of the deadtime generator and the reference signal OC*i* REF (where CC*i* P = 0, CC*i* NP = 0, MOE = 1, CC*i* E = 1, and CC*i* NE = 1 in these examples)

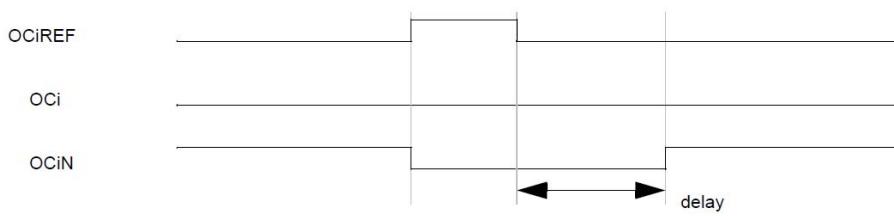
**Figure 73. Complementary output with deadtime insertion**



**Figure 74. Deadtime waveforms with a delay greater than the negative pulse**



**Figure 75. Deadtime waveforms with a delay greater than the positive pulse**



The deadtime delay is the same for each of the channels and is programmable with the DTG bits in the TIM1\_DTR register. Refer to [Section 17.7.31: Deadtime register \(TIM1\\_DTR\) on page 217](#) for delay calculation.

### Re-directing OC<sub>i</sub>REF to OC<sub>i</sub> or OC<sub>i</sub>N

In output mode (forced, output compare, or PWM), OC<sub>i</sub>REF can be re-directed to the OC<sub>i</sub> or OC<sub>i</sub>N outputs by configuring the CC<sub>i</sub>E and CC<sub>i</sub>NE bits in the corresponding TIM1\_CCER<sub>i</sub> registers. This means bypassing the deadtime generator which allows a specific waveform (such as PWM or static active level) to be sent on one output while the complementary output remains at its inactive level. Alternative possibilities are to have both outputs at inactive level or both outputs active and complementary with deadtime.

**Note:** *When only OC<sub>i</sub>N is enabled (CC<sub>i</sub>E = 0, CC<sub>i</sub>NE = 1), it is not complemented and becomes active as soon as OC<sub>i</sub>REF is high. For example, if CC<sub>i</sub>NP = 0 then OC<sub>i</sub>N = OC<sub>i</sub>REF. On the other hand, when both OC<sub>i</sub> and OC<sub>i</sub>N are enabled (CC<sub>i</sub>E = CC<sub>i</sub>NE = 1), OC<sub>i</sub> becomes active when OC<sub>i</sub>REF is high whereas OC<sub>i</sub>N is complemented and becomes active when OC<sub>i</sub>REF is low.*

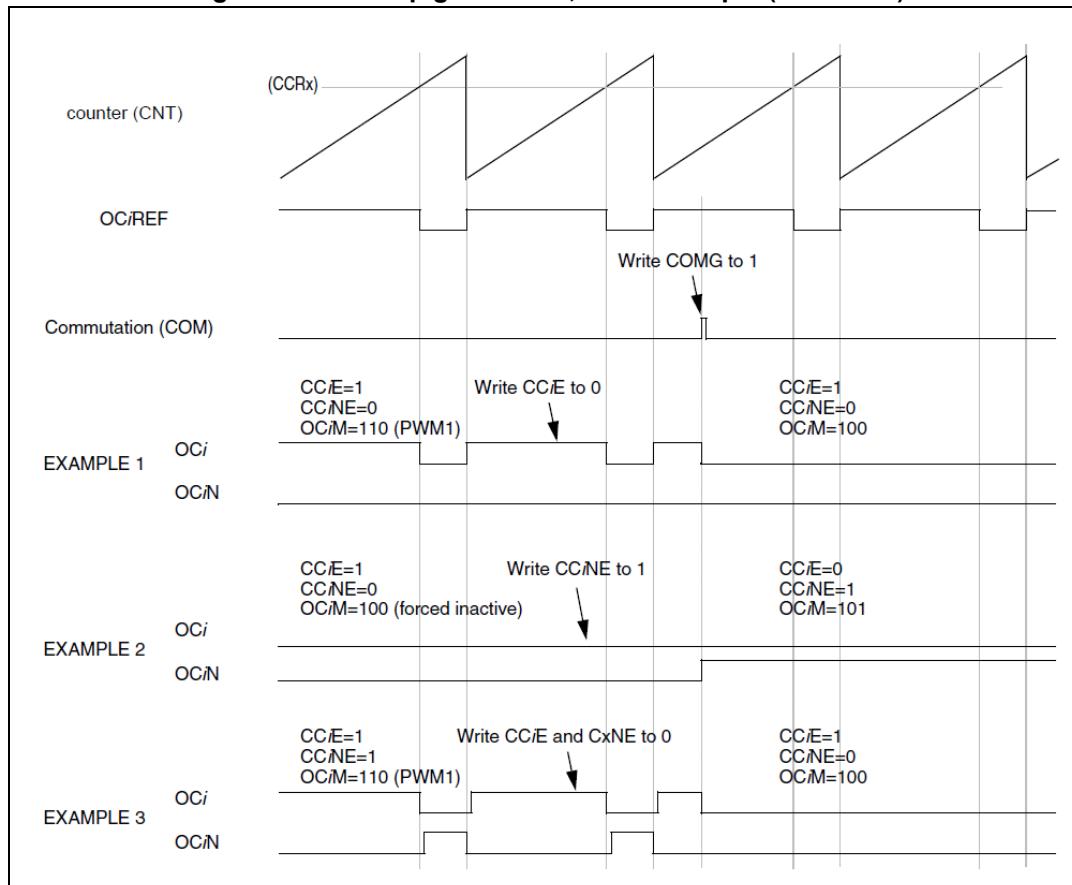
### Six-step PWM generation for motor control

When complementary outputs are implemented on a channel, preload bits are available on the OC<sub>i</sub> M, CC<sub>i</sub> E and CC<sub>i</sub> NE bits. The preload bits are transferred to the active bits at the commutation event (COM). This allows the configuration for the next step to be programmed in advance and for configuration of all the channels to be changed at the same time. The COM event can be generated by software by setting the COMG bit in the TIM1\_EGR register or by hardware trigger (on the rising edge of TRGI).

A flag is set when the COM event occurs (COMIF bit in the TIM1\_SR register) which can generate an interrupt (if the COMIE bit is set in the TIM1\_IER register).

*Figure 76* shows the behavior of the OC<sub>i</sub> and OC<sub>i</sub> N outputs when a COM event occurs, for three different examples of programmed configurations.

Figure 76. Six-step generation, COM example (OSSR = 1)



### 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 TIM1\_BKR register).

When exiting from reset, the break circuit is disabled and the MOE bit is low. The break function is enabled by setting the BKE bit in the TIM1\_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 TIM1\_BKR register). It results in some delays between the asynchronous and the synchronous signals. For example, if MOE is written to 1 after it has been low, a delay (dummy instruction) must be inserted before it can be read 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 reset state (selected by the OSS1 bit). This happens even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OIS<sub>i</sub> bits in the TIM1\_OISR register as soon as MOE = 0. If OSS1 = 0, the timer releases the enable output otherwise the enable output remains high.
- When complementary outputs are implemented:
  - The outputs are first put in inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
  - If the timer clock is still present, the deadtime generator is reactivated to drive the outputs with the level programmed in the OIS<sub>i</sub> and OIS<sub>i</sub> N bits after a deadtime. Even in this case, OC<sub>i</sub> and OC<sub>i</sub> N cannot be driven to their active level together. Note that because of the resynchronization on MOE, the deadtime duration is a bit longer than usual (around two ck\_tim clock cycles).
- The break status flag (BIF bit in the TIM1\_SR1 register) is set. An interrupt can be generated if the BIE bit in the TIM1\_IER register is set.
- If the AOE bit in the TIM1\_BKR register is set, the MOE bit is automatically set again at the next UEV. This can be used to perform a regulation. Otherwise, MOE remains low until it is written to 1 again. In this case, it can be used for security and the break input can be connected to an alarm from power drivers, thermal sensors, or any security components.

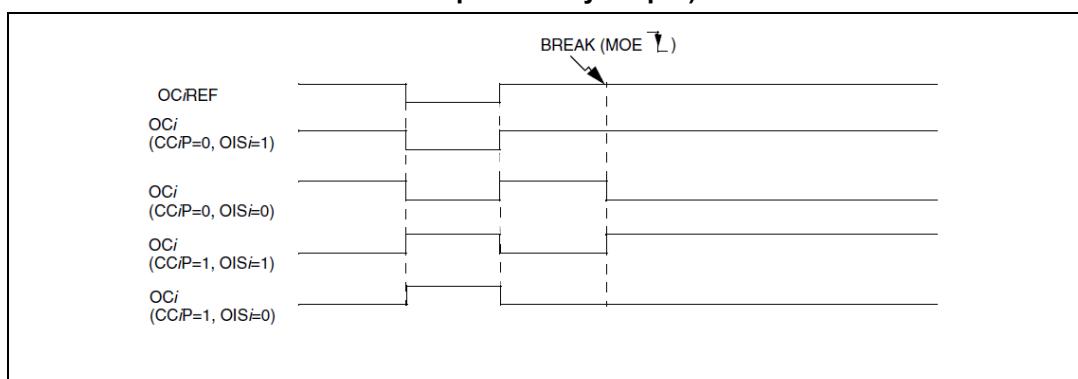
*Note:* The break inputs act on signal level. Thus, the MOE bit cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.

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 the TIM1\_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 the configuration of several parameters (OC<sub>i</sub> polarities and state when disabled, OCIM configurations, break enable, and polarity) to be frozen. Three levels of protection can be selected using the LOCK bits in the TIM1\_BKR register. The LOCK bits can be written only once after an MCU reset.

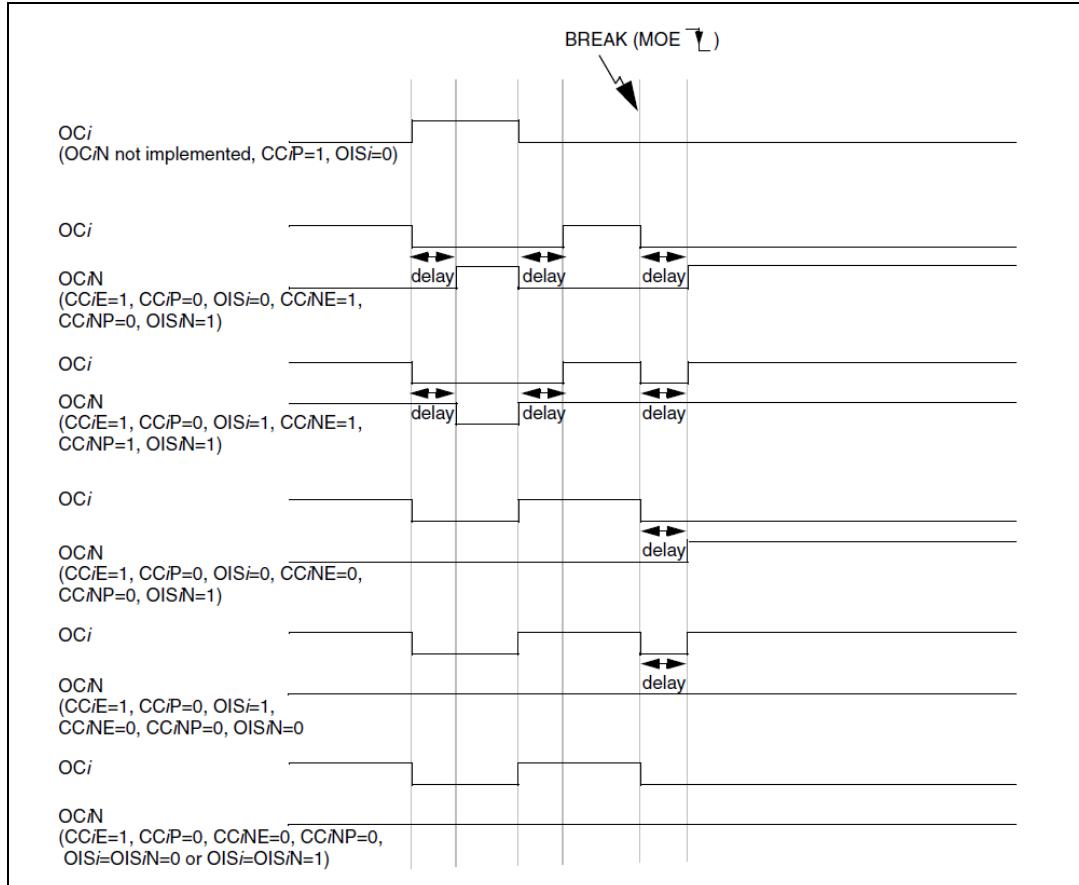
Figure 77 shows an example of the behavior of the outputs in response to a break.

**Figure 77. Behavior of outputs in response to a break (channel without complementary output)**



*Figure 78* shows an example of behavior of the complementary outputs (TIM1 only) in response to a break.

**Figure 78. Behavior of outputs in response to a break (TIM1 complementary outputs)**



### 17.5.9 Clearing the OC<sub>i</sub>REF signal on an external event

The OC<sub>i</sub>REF signal of a given channel can be cleared when a high level is detected on ETRF (if OC<sub>i</sub>CE =1 in the TIM1\_CCMR<sub>i</sub> registers, one enable bit per channel). The OC<sub>i</sub>REF signal remains low until the next UEV occurs. This function can be used in output compare mode and PWM mode only. It does not work in forced mode.

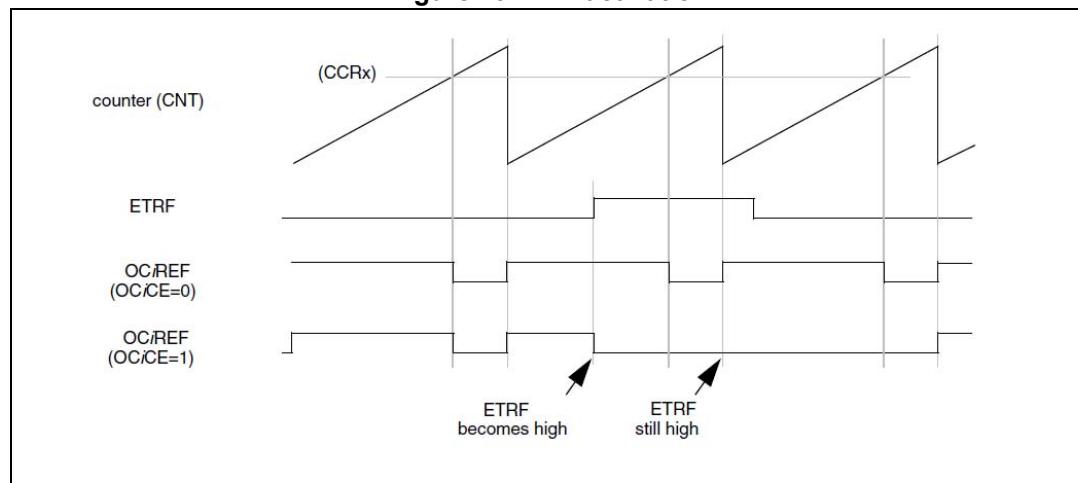
The OC<sub>i</sub>REF signal can be connected to the output of a comparator and be used for current handling by configuring the external trigger as follows:

1. Switch off the external trigger prescaler by setting bits ETPS[1:0] in the TIM1\_ETR register to 00.
2. Disable external clock mode 2 by setting the ECE bit in the TIM1\_ETR register to 0
3. Configure the external trigger polarity (ETP) and the external trigger filter (ETF) as desired.

Refer to [Figure 47: External trigger input block diagram](#).

[Figure 79](#) shows the behavior of the OC<sub>i</sub>REF signal when the ETRF input becomes high, for both values of the enable bits OC<sub>i</sub>CE. In this example, the timer is programmed in PWM mode.

**Figure 79. ETR activation**



### 17.5.10 Encoder interface mode

Encoder interface mode is typically used for motor control. It can be selected by writing:

- SMS = 001 in the TIM1\_SMCR register if the counter is counting on TI2 edges only
- SMS = 010 if the counter is counting on TI1 edges only
- SMS = 011 if the counter is counting on both TI1 and TI2 edges

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIM1\_CCER1 register. When needed, the input filter can also be programmed.

The two inputs TI1 and TI2 are used to interface an incremental encoder (see [Table 37](#)). If the counter is enabled (when the CEN bit in the TIM1\_CR1 register is written to 1), it is clocked by each valid transition on TI1FP1 or TI2FP2 (see [Figure 64: Input stage of TIM 1 channel 1](#)). The transition sequences of the two inputs (TI1 and TI2) are evaluated and generate count pulses and a direction signal. Depending on the sequence, the counter counts up or down, and the DIR bit in the TIM1\_CR1 register is modified accordingly by hardware. The DIR bit is calculated at each transition based on inputs from either TI1 or TI2, without this being dependent on whether the counter is counting pulses on TI1, TI2 or both.

Encoder interface mode acts as an external clock with direction selection. The counter counts continuously between 0 and the auto-reload value in the TIM1\_ARR register (0 to ARR or ARR down to 0 depending on the direction). TIM1\_ARR must be configured before starting. The capture, compare, prescaler, and trigger output features continue to work as normal in this mode. Encoder mode and external clock mode 2 are not compatible and must not be selected together.

In encoder interface mode, the counter is modified automatically depending on the speed and the direction of the incremental encoder. The content of the counter therefore always represents the encoder's position. The count direction corresponds to the rotation direction of the connected sensor. [Table 37](#) summarizes the possible combinations of counting directions and encoder signals, assuming that TI1 and TI2 do not switch at the same time.

**Table 37. Counting direction versus encoder signals**

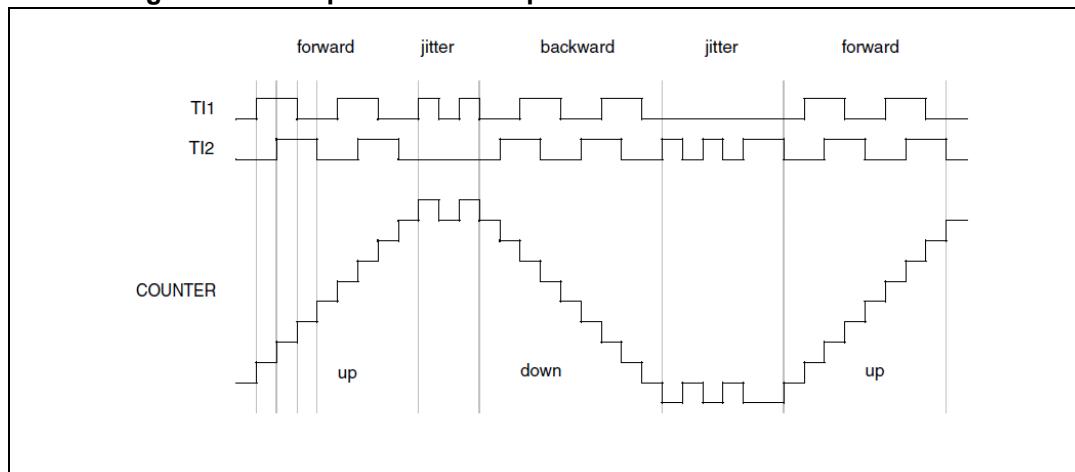
Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No count	No count
	Low	Up	Down	No count	No count
Counting on TI2 only	High	No count	No count	Up	Down
	Low	No count	No count	Down	Up
Counting on both TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators are normally used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a counter reset.

[Figure 80](#) 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 one of the switching points. In the example below, configuration is as follows:

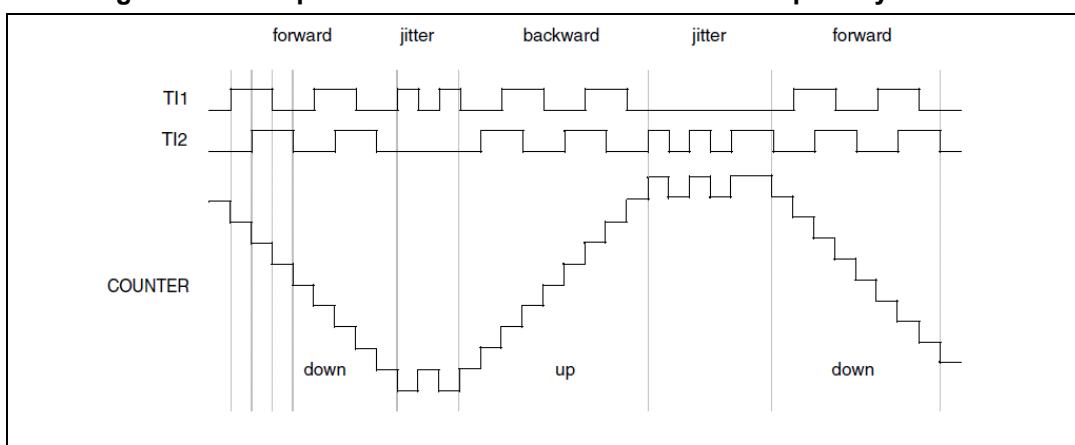
- CC1S = 01 (TIM1\_CCMR1 register, IC1 mapped on TI1)
- CC2S = 01 (TIM1\_CCMR2 register, IC2 mapped on TI2)
- CC1P = 0 (TIM1\_CCER1 register, IC1 non-inverted, IC1=TI1)
- CC2P = 0 (TIM1\_CCER2 register, IC2 non-inverted, IC2=TI2)
- SMS = 011 (TIM1\_SMCR register, both inputs are active on both rising and falling edges).
- CEN = 1 (TIM1\_CR1 register, counter is enabled)

**Figure 80. Example of counter operation in encoder interface mode**



[Figure 81](#) gives an example of counter behavior when IC1 polarity is inverted (same configuration as [Figure 80](#) except that CC1P =1).

**Figure 81. Example of encoder interface mode with IC1 polarity inverted**



When the timer is configured in encoder interface mode, it provides information on the current position of the sensors. Dynamic information, such as speed, acceleration, and slowdown, can be obtained 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 intervals. This can be done by latching the counter value into a third input capture register, if one is available. In this case, the capture signal must be periodic and can be generated by another timer.

## 17.6 TIM1 interrupts

TIM1 has eight interrupt request sources, mapped on 2 interrupt vectors:

- Break interrupt
- Trigger interrupt
- Commutation interrupt
- Capture/compare 4 interrupt
- Capture/compare 3 interrupt
- Capture/compare 2 interrupt
- Capture/compare 1 interrupt
- Update interrupt (example: overflow, underflow, and counter initialization)

To use the interrupt features for each interrupt channel used, set the desired interrupt enable bits (BIE, TIE, COMIE, CC<sub>i</sub>E, and UIE) in the TIM1\_IER register to enable interrupt requests.

The different interrupt sources can also be generated by software using the corresponding bits in the TIM1\_EGR register.

## 17.7 TIM1 registers

### 17.7.1 Control register 1 (TIM1\_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	rw

Bit 7 **ARPE**: Auto-reload preload enable

- 0: TIM1\_ARR register is not buffered through a preload register. It can be written directly
- 1: TIM1\_ARR register is buffered through a preload register

Bits 6:5 **CMS[1:0]**: Center-aligned mode selection

- 00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
- 01: Center-aligned mode 1. The counter counts up and down alternately. Output compare interrupt flags of channels configured in output (CC/S = 00 in TIM1\_CCMR<sub>i</sub> registers) 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 (CC/S = 00 in CCMR<sub>i</sub> registers) 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 (CC/S = 00 in TIM1\_CCMR<sub>i</sub> registers) are set both when the counter is counting up and down.

*Note: It is not allowed to switch from edge-aligned mode to center-aligned mode while the counter is enabled (CEN = 1)*

*Encoder mode (SMS = 001, 010 or 011 in TIM1\_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 by the UDIS bit, the UIF bit is set and an update interrupt request is sent when one of the following events occurs:

- Registers are updated (counter overflow/underflow)
- UG bit is set by software
- Update event is generated through the clock/trigger controller

1: When enabled by the UDIS bit, the UIF bit is set and an update interrupt request is sent only when registers are updated (counter overflow/underflow).

**Bit 1 UDIS:** Update disable.

0: A UEV is generated as soon as a counter overflow occurs, a software update is generated, or a hardware reset is generated by the clock/trigger mode controller. Buffered registers are then loaded with their preload values.

1: A UEV is not generated and shadow registers keep their value (ARR, PSC, CCR $i$ ). The counter and the prescaler are re-initialized if the UG bit is set or if a hardware reset is received from the clock/trigger mode controller.

**Bit 0 CEN:** Counter enable

0: Counter disabled

1: Counter enabled

*Note: External clock, trigger gated mode, and encoder mode can work only if the CEN bit has been previously set by software. However, trigger mode can set the CEN bit automatically by hardware.*

### 17.7.2 Control register 2 (TIM1\_CR2)

Address offset: 0x01

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	MMS[2:0]				Reserved	COMS	CCPC
r	rw	rw	rw			rw	rw

Bit 7 **Reserved**

Bits 6:4 **MMS[2:0]: Master mode selection**

These bits select the information to be sent in master mode to the ADC or to the other timers for synchronization (TRGO). The combination is as follows:

000: Reset - The UG bit from the TIM1\_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in reset mode), 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 or the ADC to control a window in which a slave timer or the ADC is enabled. The counter enable signal is generated by a logic OR between the CEN control bit and the trigger input when configured in trigger 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 TIM1\_SMCR register).

010: Update - The update event is selected as trigger output (TRGO)

011: Compare pulse (MATCH1) - The trigger output sends a positive pulse when the CC1IF flag is to be set (even if it was already high), as soon as a capture or a compare match occurs (TRGO).

100: Compare - OC1REF signal is used as trigger output (TRGO)

101: Compare - OC2REF signal is used as trigger output (TRGO)

110: Compare - OC3REF signal is used as trigger output (TRGO)

111: Compare - OC4REF signal is used as trigger output (TRGO)

Bit3 Reserved, must be kept cleared.

Bit 2 **COMS: Capture/compare control update selection**

0: When capture/compare control bits are preloaded (CCPC = 1), they are updated by setting the COMG bit.

1: When capture/compare control bits are preloaded (CCPC = 1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI.

*Note: This bit acts only on channels with complementary outputs.*

Bit 1 Reserved, forced by hardware to 0

Bit 0 **CCPC: Capture/compare preloaded control**

0: The CCiE, CCiNE, CCiP, and CCiNP bits in the TIM1\_CCER*i* registers and the OCiM bit in the TIM1\_CCMR*i* registers are not preloaded

1: CCiE, CCiNE, CCiP, CCiNP and OCiM bits are preloaded, after having been written, they are updated only when COMG bit is set in the TIM1\_EGR register.

*Note: This bit acts only on channels with complementary outputs.*

### 17.7.3 Slave mode control register (TIM1\_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	rw	r	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 TIM1 and another timer (through TRGO).

Bits 6:4 **TS[2:0]**: Trigger selection

This bit field selects the trigger input (TRGI) to be used to synchronize the counter.

000: Internal trigger ITR0 connected to TIM6 TRGO (\*)

001: Reserved

010: Reserved

011: Internal trigger ITR3 connected to TIM5 TRGO (\*)

100: TI1 edge detector (TI1F\_ED)

101: Filtered timer input 1 (TI1FP1)

110: Filtered timer input 2 (TI2FP2)

111: External trigger input (ETRF)

(\*): 8K low-density devices only otherwise reserved.

*Note: These bits must only be changed when they are not used (e.g. when SMS = 000) to avoid incorrect 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, the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up or down on TI2FP2 edge depending on TI1FP1 level

010: Encoder mode 2 - Counter counts up or down on TI1FP1 edge depending on TI2FP2 level

011: Encoder mode 3 - Counter counts up or down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset mode - Rising edge of the selected trigger signal (TRGI) re-initializes the counter and generates an update of the registers.

101: Trigger 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 standard mode - The counter starts at a rising edge of the trigger TRGI (but, it is not reset). Only the start of the counter is controlled.

111: External clock mode 1 - Rising edges of the selected trigger (TRGI) clock the counter

*Note: Trigger gated mode must not be used if TI1F\_ED is selected as the trigger input (TS = 100). TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas trigger gated mode checks the level of the trigger signal.*

### 17.7.4 External trigger register (TIM1\_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	rw	rw	rw	rw

Bit 7 **ETP**: External trigger polarity

This bit selects whether ETR or  $\overline{\text{ETR}}$  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 external clock mode 1 with TRGI connected to ETRF (SMS = 111 and TS = 111 in the TIM1\_SMCR register).*

*It is possible to simultaneously use 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 these cases (TS bits must not be 111 in the TIM1\_SMCR register).*

*If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.*

**Bits 5:4 ETPS:** External trigger prescaler

The ETRP frequency must be, at most, 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 bitfield defines the frequency used to sample the ETRP signal and the length of the digital filter applied to it. 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 (TIM1\_IER)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
rw	rw	rw	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

Bit 5 **COMIE**: Commutation interrupt enable

- 0: Commutation interrupt disabled
- 1: Commutation interrupt enabled

Bit 4 **CC4IE**: Capture/compare 4 interrupt enable

- 0: CC4 interrupt disabled
- 1: CC4 interrupt enabled

Bit 3 **CC3IE**: Capture/compare 3 interrupt enable

- 0: CC3 interrupt disabled
- 1: CC3 interrupt enabled

Bit 2 **CC2IE**: Capture/compare 2 interrupt enable

- 0: CC2 interrupt disabled
- 1: CC2 interrupt enabled

Bit 1 **CC1IE**: Capture/compare 1 interrupt enable

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled
- 1: Update interrupt enabled

### 17.7.6 Status register 1 (TIM1\_SR1)

Address offset: 0x05

Reset value: 0x00

7	6	5	4	3	2	1	0
BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
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 has 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 a trigger event (the active edge is detected on a TRGI signal, both edges are detected if trigger gated mode is selected). It is cleared by software.

0: No trigger event has occurred

1: Trigger interrupt pending

Bit 5 **COMIF**: Commutation interrupt flag

This flag is set by hardware on a COM (when capture/compare control bits - CC*i*E, CC*i*NE, OC*i*M - have been updated). It is cleared by software.

0: No COM has occurred

1: COM interrupt pending

Bit 4 **CC4IF**: Capture/compare 4 interrupt flag

Refer to CC1IF description

Bit 3 **CC3IF**: Capture/compare 3 interrupt flag

Refer to CC1IF description

Bit 2 **CC2IF**: Capture/compare 2 interrupt flag

Refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

**If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits from TIM1\_CR1 register description). It is cleared by software.

0: No match

1: The content of the counter register TIM1\_CNT matches the content of the TIM1\_CCR1 register

*Note: When the contents of TIMx\_CCRi are greater than the contents of TIMx\_ARR, the CCiIF bit goes high on the counter overflow (in up-counting and up/down-counting modes) or underflow (in down-counting mode)*

**If channel CC1 is configured as input:**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIM1\_CCR1L register.

0: No input capture has occurred

1: The counter value has been captured in the TIM1\_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 has occurred

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow if UDIS = 0 in the TIM1\_CR1 register
- When CNT is re-initialized by software using the UG bit in TIM1\_EGR register, if URS = 0 and UDIS = 0 in the TIM1\_CR1 register.
- When CNT is re-initialized by a trigger event (refer to the TIM1\_SMCR register description), if URS = 0 and UDIS = 0 in the TIM1\_CR1 register.

### 17.7.7 Status register 2 (TIM1\_SR2)

Address offset: 0x06

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved		CC4OF	CC3OF	CC2OF	CC1OF	Reserved	
r		rc_w0	rc_w0	rc_w0	rc_w0	r	

Bits 7:5 Reserved, must be kept cleared

Bit 4 **CC4OF**: Capture/compare 4 overcapture flag

Refer to CC1OF description

Bit 3 **CC3OF**: Capture/compare 3 overcapture flag

Refer to CC1OF description

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 TIM1\_CCR1 register while CC1IF flag was already set

Bit 0 Reserved, must be kept cleared.

### 17.7.8 Event generation register (TIM1\_EGR)

Address offset: 0x07

Reset value: 0x00

7	6	5	4	3	2	1	0
BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG
w	w	w	w	w	w	w	w

Bit 7 **BG**: Break generation  
This bit is set by software to generate an event. It is automatically cleared by hardware.  
0: No action  
1: A break event is generated. The MOE bit is cleared and the 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 to generate an event. It is automatically cleared by hardware.  
0: No action  
1: The TIF flag is set in TIM1\_SR1 register. An interrupt is generated if enabled by the TIE bit.

Bit 5 **COMG**: Capture/compare control update generation  
This bit can be set by software and is automatically cleared by hardware.  
0: No action  
1: When the CCPC bit in the TIM1\_CR2 register is set, it allows the CC/E, CC/NE CC/P, CC/NP, and OC/M bits to be updated.

*Note: This bit acts only on channels that have a complementary output.*

Bit 4 **CC4G**: Capture/compare 4 generation  
Refer to CC1G description.

Bit 3 **CC3G**: Capture/compare 3 generation  
Refer to CC1G description.

Bit 2 **CC2G**: Capture/compare 2 generation

Refer to CC1G description.

Bit 1 **CC1G**: Capture/compare 1 generation.

This bit is set by software 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:

The CC1IF flag is set and the corresponding interrupt request is sent if enabled.

If the CC1 channel is configured in input mode:

The current value of the counter is captured in the TIM1\_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 is already high.

Bit 0 **UG**: Update generation

This bit can be set by software and 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 also cleared. The counter is cleared if center-aligned mode is selected or if DIR = 0 (up-counting). Otherwise, it takes the auto-reload value (TIM1\_ARR) if DIR = 1 (down-counting).

### 17.7.9 Capture/compare mode register 1 (TIM1\_CCMR1)

Address offset: 0x08

Reset value: 0x00

This 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 output mode. For a given bit, OC $i$ i describes its function when the channel is configured in output, IC $i$ i describes its function when the channel is configured in input. Therefore, be aware that the same bit can have a different meaning for the input and output stage.

### Channel configured in output

7	6	5	4	3	2	1	0
OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **OC1CE**: Output compare 1 clear enable

This bit is used to enable the clearing of the channel 1 output compare signal (OC1REF) by an external event on the TIM1\_ETR pin (see [Section 17.5.9 on page 184](#)).

0: OC1REF is not affected by the ETRF input signal (derived from the TIM1\_ETR pin)  
1: OC1REF is cleared as soon as a high level is detected on ETRF input signal (derived from the TIM1\_ETR pin).

Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal, OC1REF, from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on the CC1P bit.

000: Frozen - The comparison between the output compare register TIM1\_CCR1 and the counter register TIM1\_CNT has no effect on the outputs.

001: Set channel 1 to active level on match - OC1REF signal is forced high when the counter register TIM1\_CNT matches the capture/compare register 1 (TIM1\_CCR1).

010: Set channel 1 to inactive level on match - OC1REF signal is forced low when the counter register TIM1\_CNT matches the capture/compare register 1 (TIM1\_CCR1).

011: Toggle - OC1REF toggles when TIM1\_CNT = TIM1\_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 TIM1\_CNT < TIM1\_CCR1, otherwise, the channel is inactive. In down-counting, channel 1 is inactive (OC1REF = 0) as long as TIM1\_CNT > TIM1\_CCR1, otherwise, the channel is active (OC1REF = 1).

111: PWM mode 2 - In up-counting, channel 1 is inactive as long as TIM1\_CNT < TIM1\_CCR1, otherwise, the channel is active. In down-counting, channel 1 is active as long as TIM1\_CNT > TIM1\_CCR1, otherwise, the channel is inactive.

*Note: These bits can no longer be modified while LOCK level 3 has been programmed (LOCK bits in TIM1\_BKR register) and CC1S = 00 (the channel is configured in output).*

*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 [PWM mode on page 174](#) for more details).*

*On channels that have a complementary output, this bitfield is preloaded. If the CCPC bit is set in the TIM1\_CR2 register, the OCM active bits take the new value from the preload bits only when a COM is generated.*

**Bit 3 OC1PE:** Output compare 1 preload enable

0: Preload register on TIM1\_CCR1 disabled. TIM1\_CCR1 can be written at anytime. The new value is taken into account immediately.

1: Preload register on TIM1\_CCR1 enabled. Read/write operations access the preload register. TIM1\_CCR1 preload value is loaded in the shadow register at each UEV.

*Note: These bits can no longer be modified while LOCK level 3 has been programmed (LOCK bits in TIM1\_BKR register) and CC1S = 00 (the channel is configured in output).*

*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 TIM1\_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 the 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 the CC1 output. If this happens, OC is set to the compare level irrespective of the result of the comparison. The delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

**Bits 1:0 CC1S[1:0]:** Capture/compare 1 selection

This bitfield 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: CC1 channel is configured as input, IC1 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIM1\_SMCR register).

*Note: CC1S bits are writable only when the channel is off (CC1E = 0 in TIM1\_CCER1).*

### Channel configured in input

7	6	5	4	3	2	1	0
IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw

#### Bits 7:4 IC1F[3:0]: Input capture 1 filter

This bitfield defines  $f_{\text{SAMPLING}}$ , 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,  $f_{\text{SAMPLING}} = f_{\text{MASTER}}$
- 0001:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}, N = 2$
- 0010:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}, N = 4$
- 0011:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}, N = 8$
- 0100:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/2, N = 6$
- 0101:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/2, N = 8$
- 0110:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/4, N = 6$
- 0111:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/4, N = 8$
- 1000:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/8, N = 6$
- 1001:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/8, N = 8$
- 1010:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/16, N = 5$
- 1011:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/16, N = 6$
- 1100:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/16, N = 8$
- 1101:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/32, N = 5$
- 1110:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/32, N = 6$
- 1111:  $f_{\text{SAMPLING}} = f_{\text{MASTER}}/32, N = 8$

*Note: Even on channels that have a complementary output, this bit field is not preloaded and does not take into account the content of the CCPC bit (in the TIM1\_CR2 register).*

#### Bits 3:2 IC1PSC[1:0]: Input capture 1 prescaler

This bitfield defines the ratio of the prescaler acting on CC1 input (IC1). The prescaler is reset as soon as CC1E = 0 (TIM1\_CCER register).

- 00: No prescaler, capture is made each time an edge is detected on the capture input
- 01: Capture is made once every 2 events
- 10: Capture is made once every 4 events
- 11: Capture is made once every 8 events

#### Bits 1:0 CC1S[1:0]: Capture/compare 1 selection

This bitfield defines the direction of the channel (input/output) and 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: CC1 channel is configured as input, IC1 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIM1\_SMCR register).

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIM1\_CCER1).*

### 17.7.10 Capture/compare mode register 2 (TIM1\_CCMR2)

Address offset: 0x09

Reset value: 0x00

#### Channel configured in output

7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]				OC2PE	OC2FE	CC2S[1:0]
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **OC2CE**: Output compare 2 clear enable

Bits 6:4 **OC2M(2:0)**: Output compare 2 mode

Bit 3 **OC2PE**: Output compare 2 preload enable

Bit 2 **OC2FE**: Output compare 2 fast enable

Bits 1:0 **CC2S[1:0]**: Capture/compare 2 selection

This bitfield defines the direction of the channel (input/output) and 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 and CC2NE = 0 and updated in TIM1\_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	rw	rw	rw	rw	rw

Bits 7:4 **IC2F**: Input capture 2 filter

Bits 3:2 **IC2PSC(1:0)**: Input capture 2 prescaler

Bits 1:0 **CC2S[1:0]**: Capture/compare 2 selection

This bitfield defines the direction of the channel (input/output) and 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: CC2 channel is configured as input, IC2 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIM1\_SMCR register).

*Note:* CC2S bits are writable only when the channel is off (CC2E and CC2NE = 0 and updated in TIM1\_CCER1).

### 17.7.11 Capture/compare mode register 3 (TIM1\_CCMR3)

Address offset: 0x0A

Reset value: 0x00

Refer to the CCMR1 register description above.

#### Channel configured in output

	7	6	5	4	3	2	1	0
	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **OC3CE**: Output compare 3 clear enable

Bits 6:4 **OC3M[2:0]**: Output compare 3 mode

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 **OC3FE**: Output compare 3 fast enable

Bits 1:0 **CC3S[1:0]**: Capture/compare 3 selection

This bitfield defines the direction of the channel (input/output) and the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3FP3

10: CC3 channel is configured as input, IC3 is mapped on TI4FP3

11: Reserved

*Note:* CC3S bits are writable only when the channel is off (CC3E and CC3NE = 0 and updated in TIM1\_CCER2).

#### Channel configured in input

	7	6	5	4	3	2	1	0
	IC3F[3:0]			IC3PSC[1:0]		CC3S[1:0]		
		rw	rw	rw	rw	rw	rw	rw

Bits 7:4 **IC3F**: Input capture 3 filter

Bits 3:2 **IC3PSC[1:0]**: Input capture 3 prescaler

Bits 1:0 **CC3S[1:0]**: Capture/compare 3 selection

This bitfield defines the direction of the channel (input/output) and the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3FP3

10: CC3 channel is configured as input, IC3 is mapped on TI4FP3

11: Reserved

CC3S bits are writable only when the channel is off (CC3E and CC3NE = 0 and updated in TIM1\_CCER2).

### 17.7.12 Capture/compare mode register 4 (TIM1\_CCMR4)

Address offset: 0xB

Reset value: 0x00

Refer to the CCMR1 register description above.

#### Channel configured in output

	7	6	5	4	3	2	1	0
	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	
	rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **OC4CE**: Output compare 4 clear enable

Bits 6:4 **OC4M[2:0]**: Output compare 4 mode

Bit 3 **OC4PE**: Output compare 4 preload enable

Bit 2 **OC4FE**: Output compare 4 fast enable

Bits 1:0 **CC4S[1:0]**: Capture/compare 4 selection

This bitfield defines the direction of the channel (input/output) and the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4FP4

10: CC4 channel is configured as input, IC4 is mapped on TI3FP4

11: Reserved

*Note: CC4S bits are writable only when the channel is off (CC4E and CC4NE = 0 and updated in TIM1\_CCER2).*

#### Channel configured in input

	7	6	5	4	3	2	1	0
	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:4 **IC4F**: Input capture 4 filter

Bits 3:2 **IC4PSC[1:0]**: Input capture 4 prescaler

Bits 1:0 **CC4S[1:0]**: Capture/compare 4 selection

This bitfield defines the direction of the channel (input/output) and the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4FP4

10: CC4 channel is configured as input, IC4 is mapped on TI3FP4

11: Reserved

*Note: CC4S bits are writable only when the channel is off (CC4E and CC4NE = 0 and updated in TIM1\_CCER2).*

### 17.7.13 Capture/compare enable register 1 (TIM1\_CCER1)

Address offset: 0x0C

Reset value: 0x00

7	6	5	4	3	2	1	0
CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **CC2NP**: Capture/compare 2 complementary output polarity  
Refer to CC1NP description.

Bit 6 **CC2NE**: Capture/compare 2 complementary output enable  
Refer to CC1NE description.

Bit 5 **CC2P**: Capture/compare 2 output polarity  
Refer to CC1P description.

Bit 4 **CC2E**: Capture/compare 2 output enable  
Refer to CC1E description.

Bit 3 **CC1NP**: Capture/compare 1 complementary output polarity  
0: OC1N active high  
1: OC1N active low

*Note: This bit is no longer writable while LOCK level 2 or 3 have been programmed (LOCK bits in TIM1\_BKR register) and CC1S = 00 (the channel is configured in output).*

*On channels that have a complementary output, this bit is preloaded. If the CCPCE bit is set in the TIM1\_CR2 register, the CC1NP active bit takes the new value from the preload bit only when a COM is generated.*

Bit 2 **CC1NE**: Capture/compare 1 complementary output enable

0: Off - OC1N is not active. OC1N level is then a function of the MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

1: On - OC1N signal is output on the corresponding output pin depending on the MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

*Note: On channels that have a complementary output, this bit is preloaded. If the CCPC bit is set in the TIM1\_CR2 register, the CC1NE active bit takes the new value from the preload bit when a COM is generated.*

Bit 1 **CC1P**: Capture/compare 1 output polarity

**CC1 channel configured as output:**

0: OC1 active high

1: OC1 active low

**CC1 channel configured as input for trigger function (see Figure 64):**

0: Trigger on a high level or rising edge of TI1F

1: Trigger on a low level or falling edge of TI1F

**CC1 channel configured as input for capture function (see Figure 64):**

0: Capture on a rising edge of TI1F or TI2F

1: Capture on a falling edge of TI1F or TI2F

*Note: This bit is no longer writable while LOCK level 2 or 3 have been programmed (LOCK bits in TIM1\_BKR register).*

*On channels that have a complementary output, this bit is preloaded. If the CCPC bit is set in the TIM1\_CR2 register, the CC1P active bit takes the new value from the preload bit when a COM is generated.*

Bit 0 **CC1E**: Capture/compare 1 output enable

**CC1 channel is configured as output:**

0: Off - OC1 is not active. OC1 level is then a function of the MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

1: On - OC1 signal is output on the corresponding output pin depending on the MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

**CC1 channel is configured as input:**

This bit determines if a capture of the counter value can be made in the input capture/compare register 1 (TIM1\_CCR1) or not.

0: Capture disabled

1: Capture enabled

*Note: On channels that have a complementary output, this bit is preloaded. If the CCPC bit is set in the TIM1\_CR2 register, the CC1E active bit takes the new value from the preload bit only when a COM is generated.*

**Table 38. Output control for complementary OC*i* and OC*iN* channels with break feature<sup>(1)</sup>**

Control bits					Output states	
MOE	OSSI	OSSR	CCiE	CCiNE	OC <i>i</i>	OC <i>iN</i>
1	x <sup>(2)</sup>	0	0	0	Output disabled (not driven by the timer)	Output disabled (not driven by the timer)
		0	0	1	Output disabled (not driven by the timer)	OC <i>i</i> REF + polarity OC <i>iN</i> = OC <i>i</i> REF xor CC <i>i</i> NP
		0	1	0	OC <i>i</i> REF + polarity OC <i>i</i> = OC <i>i</i> REF xor CC <i>i</i> P	Output disabled (not driven by the timer)
		0	1	1	OC <i>i</i> REF + polarity + deadtime	Complementary to OC <i>i</i> REF (not OC <i>i</i> REF) + polarity + deadtime
		1	0	0	Output disabled (not driven by the timer)	Output disabled (not driven by the timer)
		1	0	1	Off state (output enabled with inactive state) OC <i>i</i> = CC <i>i</i> P	OC <i>i</i> REF + polarity OC <i>iN</i> = OC <i>i</i> REF xor CC <i>i</i> NP
		1	1	0	OC <i>i</i> REF + polarity OC <i>i</i> = OC <i>i</i> REF xor CC <i>i</i> P	Off state (output enabled with inactive state) OC <i>N</i> = CC <i>i</i> NP
		1	1	1	OC <i>i</i> REF + polarity + deadtime	Complementary to OC <i>i</i> REF (not OC <i>i</i> REF) + polarity + deadtime
0	x <sup>(2)</sup>	0	x	x	Output disabled (not driven by the timer)	
		0			Off state (output enabled with inactive state) Asynchronously: OC <i>i</i> = CC <i>i</i> P and OC <i>N</i> = CC <i>i</i> NP	
		0			Then if the clock is present: OC <i>i</i> = OIS <i>i</i> and OC <i>N</i> = OIS <i>N</i> after a deadtime, assuming that OIS <i>i</i> and OIS <i>N</i> do not correspond with OC <i>i</i> and OC <i>N</i> in active state	
		1				
		1				
		1				
		1				
		1				

1. Never set CCiE = CCNiE = 0 when the channel is used. When the channel is not used, program CCiP = CCiNP = OISi = OISiN = 0 otherwise.

2. Don't care

**Note:** The state of the external I/O pins connected to the OC*i* channels depends on the OC*i* channel state and the GPIO registers.

### 17.7.14 Capture/compare enable register 2 (TIM1\_CCER2)

Address offset: 0x0D

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	
r	rw	rw	rw	rw	rw	rw	rw

Bits 7:6 Reserved

Bit 5 **CC4P**: Capture/compare 4 output polarity

Refer to CC1P description.

Bit 4 **CC4E**: Capture/compare 4 output enable

Refer to CC1E description.

Bit 3 **CC3NP**: Capture/compare 3 complementary output polarity

Refer to CC1NP description.

Bit 2 **CC3NE**: Capture/compare 3 complementary output enable

Refer to CC1NE description.

Bit 1 **CC3P**: Capture/compare 3 output polarity

Refer to CC1P description.

Bit 0 **CC3E**: Capture/compare 3 output enable

Refer to CC1E description.

### 17.7.15 Counter high (TIM1\_CNTRH)

Address offset: 0x0E

Reset value: 0x00

7	6	5	4	3	2	1	0
CNT[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CNT[15:8]**: Counter value (MSB)

### 17.7.16 Counter low (TIM1\_CNTRL)

Address offset: 0x0F

Reset value: 0x00

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

Bits 7:0 **CNT[7:0]**: Counter value (LSB).

### 17.7.17 Prescaler high (TIM1\_PSCRH)

Address offset: 0x10

Reset value: 0x00

7	6	5	4	3	2	1	0
PSC[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

**PSC[15:8]**: Prescaler value (MSB)

Bits 7:0 The prescaler value divides the CK\_PSC clock frequency. The counter clock frequency  $f_{CK\_CNT}$  is equal to  $f_{CK\_PSC} / (PSCR[15:0]+1)$ . PSCR contain the value which is loaded in the active prescaler register at each UEV (including when the counter is cleared through the UG bit of the TIM1\_EGR register or through the trigger controller when configured in trigger reset mode). A UEV must be generated so that a new prescaler value can be taken into account.

### 17.7.18 Prescaler low (TIM1\_PSCRL)

Address offset: 0x11

Reset value: 0x00

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

Bits 7:0 **PSC[7:0]**: Prescaler value (LSB)

The prescaler value divides the CK\_PSC clock frequency. The counter clock frequency  $f_{CK\_CNT}$  is equal to  $f_{CK\_PSC} / (PSCR[15:0]+1)$ . PSCR contains the value which is loaded in the active prescaler register at each UEV (including when the counter is cleared through the UG bit of the TIM1\_EGR register or through the trigger controller when configured in trigger reset mode).

A UEV must be generated so that a new prescaler value can be taken into account.

### 17.7.19 Auto-reload register high (TIM1\_ARRH)

Address offset: 0x12

Reset value: 0xFF

7	6	5	4	3	2	1	0
ARR[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **ARR[15:8]**: Auto-reload value (MSB)

ARR is the value to be loaded in the actual auto-reload register. Refer to the [Section 17.3: TIM1 time base unit on page 141](#) for more details about ARR update and behavior. The counter is blocked while the auto-reload value is null.

### 17.7.20 Auto-reload register low (TIM1\_ARRL)

Address offset: 0x13

Reset value: 0xFF

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

Bits 7:0 **ARR[7:0]**: Auto-reload value (LSB).

### 17.7.21 Repetition counter register (TIM1\_RCR)

Address offset: 0x14

Reset value: 0x00

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

Bits 7:0 **REP[7:0]**: Repetition counter value.

When the preload registers are enabled, these bits allow the user to set up the update rate of the compare registers (periodic transfers from preload to shadow registers) as well as the update interrupt generation rate if the update interrupt is enabled (UIE=1).

Each time the REP\_CNT related down-counter reaches zero, a UEV is generated and it restarts counting from the REP value. As REP\_CNT is reloaded with the REP value only at the repetition update event U\_RC, any write to the TIM1\_RCR register is not taken into account until the next repetition update event.

In PWM mode (REP+1) corresponds to:

- The number of PWM periods in edge-aligned mode
- The number of half PWM periods in center-aligned mode

### 17.7.22 Capture/compare register 1 high (TIM1\_CCR1H)

Address offset: 0x15

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR1[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR1[15:8]**: Capture/compare 1 value (MSB)

If the CC1 channel is configured as output (CC1S bits in TIM1\_CCMR1 register):

The value of CCR1 is loaded permanently into the actual capture/compare 1 register if the preload feature is enabled (OC1PE bit in TIMx\_CCMR1). Otherwise, the preload value is copied in the active capture/compare 1 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIMx\_CNT, and signalled on the OC1 output.

**If the CC1 channel is configured as input (CC1S bits in TIM1\_CCMR1 register):**

The value of CCR1 is the counter value transferred by the last input capture 1 event (IC1). In this case, these bits are read only.

### 17.7.23 Capture/compare register 1 low (TIM1\_CCR1L)

Address offset: 0x16

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR1[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR1[7:0]**: Capture/compare 1 value (LSB)

### 17.7.24 Capture/compare register 2 high (TIM1\_CCR2H)

Address offset: 0x17

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR2[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR2[15:8]**: Capture/compare 2 value (MSB)

If the CC2 channel is configured as output (CC2S bits in TIM1\_CCMR2 register):

The value of CCR2 is loaded permanently into the actual capture/compare 2 register if the preload feature is not enabled (OC2PE bit in TIM1\_CCMR2). Otherwise, the preload value is copied in the active capture/compare 2 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIM1\_CNT, and signalled on the OC2 output.

If the CC2 channel is configured as input (CC2S bits in TIM1\_CCMR2 register):

The value of CCR2 is the counter value transferred by the last input capture 2 event (IC2). In this case, these bits are read only.

### 17.7.25 Capture/compare register 2 low (TIM1\_CCR2L)

Address offset: 0x18

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR2[7:0]**: Capture/compare value (LSB)

### 17.7.26 Capture/compare register 3 high (TIM1\_CCR3H)

Address offset: 0x19

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR3[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR3[15:8]**: Capture/compare value (MSB)

**If the CC3 channel is configured as output (CC3S bits in TIM1\_CCMR3 register):**

The value of CCR3 is loaded permanently into the actual capture/compare 3 register if the preload feature is not enabled (OC3PE bit in TIM1\_CCMR3). Otherwise, the preload value is copied in the active capture/compare 3 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIM1\_CNT, and signalled on the OC3 output.

**If the CC3 channel is configured as input (CC3S bits in TIM1\_CCMR3 register):**

The value of CCR3 is the counter value transferred by the last input capture 3 event (IC31).

### 17.7.27 Capture/compare register 3 low (TIM1\_CCR3L)

Address offset: 0x1A

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR3[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR3[7:0]**: Capture/compare value (LSB)

### 17.7.28 Capture/compare register 4 high (TIM1\_CCR4H)

Address offset: 0x1B

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR4[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR4[15:8]**: Capture/compare value (MSB)

**If the CC4 channel is configured as output (CC4S bits in TIM1\_CCMR4 register):**

The value of CCR4 is loaded permanently into the actual capture/compare 4 register if the preload feature is not enabled (OC4PE bit in TIM1\_CCMR4). Otherwise, the preload value is copied in the active capture/compare 4 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIM1\_CNT, and signalled on the OC4 output.

**If the CC4 channel is configured as input (CC4S bits in TIM1\_CCMR4 register):**

The value of CCR4 is the counter value transferred by the last input capture 4 event (IC4).

### 17.7.29 Capture/compare register 4 low (TIM1\_CCR4L)

Address offset: 0x1C

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR4[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR4[7:0]**: Capture/compare value (LSB)

### 17.7.30 Break register (TIM1\_BKR)

Address offset: 0x1D

Reset value: 0x00

7	6	5	4	3	2	1	0
LOCK							
MOE	AOE	BKP	BKE	OSSR	OSSI		
rw	rw	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 acts only on the channels which are configured in output.

0: OC and OCN outputs are disabled or forced to idle state

1: OC and OCN outputs are enabled if their respective enable bits are set (CC*i*E in TIM1\_CCER*i* registers).

See OC/OCN enable description for more details ([Section 17.7.13 on page 205](#)).

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 UEV (if the break input is not active)

*Note:* This bit can no longer be modified while LOCK level 1 has been programmed (LOCK bits in the TIM1\_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 while LOCK level 1 has been programmed (LOCK bits in the TIM1\_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 while LOCK level 1 has been programmed (LOCK bits in the TIM1\_BKR register).

**Bit 3 OSSR:** Off state selection for Run mode

This bit is used when MOE = 1 on channels with a complementary output which are configured as outputs. See OC/OCN enable description for more details ([Section 17.7.13](#)).

0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal = 0)

1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCiE = 1 or CCiNE = 1, after which the OC/OCN enable output signal = 1

*Note: This bit can no longer be modified while LOCK level 2 has been programmed (LOCK bits in TIM1\_BKR register).*

**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 ([Section 17.7.13](#)).

0: When inactive, OC*i* outputs are disabled (OC*i* enable output signal = 0)

1: When inactive, OC*i* outputs are forced first with their idle level as soon as CC*i*E = 1 (OC enable output signal = 1)

*Note: This bit can no longer be modified while LOCK level 2 has been programmed (LOCK bits in the TIM1\_BKR register).*

**Bits 1:0 LOCK[1:0]:** Lock configuration

These bits offer a write protection against software errors.

00: LOCK off - No bits are write protected

01: LOCK level 1 - OIS*i* bit in TIM1\_OISR register and BKE/BKP/AOE bits in TIM1\_BKR register can no longer be written.

10: LOCK level 2 - LOCK level 1 + CC polarity bits (CC*i*P bits in TIM1\_CCER*i* registers, as long as the related channel is configured in output through the CC*S* bits) as well as the OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 - LOCK level 2 + CC control bits (OC*i*M and OC*i*PE bits in TIM1\_CCMR*i* registers, as long as the related channel is configured in output through the CC*S* bits) can no longer be written.

*Note: The LOCK bits can be written only once after reset. Once the TIM1\_BKR register has been written, their content is frozen until the next reset.*

**Note:** As the bits AOE, BKP, BKE, OSSR, and OSSI can be write-locked depending on the LOCK configuration, it is necessary to configure all of them during the first write access to the TIM1\_BKR register.

### 17.7.31 Deadtime register (TIM1\_DTR)

Address offset: 0x1E

Reset value: 0x00

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

Bits 7:0 **DTG[7:0]**: Deadtime generator set-up

This bitfield defines the duration of the deadtime inserted between the complementary outputs. DT corresponds to this duration.  $t_{CK\_PSC}$  is the TIM1 clock pulse.

$DTG[7:5] = 0xx \Rightarrow DT = DTG[7:0] \times t_{dtg}$  with  $t_{dtg} = t_{CK\_PSC}$  (f1)

$DTG[7:5] = 10x \Rightarrow DT = (64 + DTG[5:0]) \times t_{dtg}$  with  $t_{dtg} = 2 \times t_{CK\_PSC}$  (f2)

$DTG[7:5] = 110 \Rightarrow DT = (32 + DTG[4:0]) \times t_{dtg}$  with  $t_{dtg} = 8 \times t_{CK\_PSC}$  (f3)

$DTG[7:5] = 111 \Rightarrow DT = (32 + DTG[4:0]) \times t_{dtg}$  with  $t_{dtg} = 16 \times t_{CK\_PSC}$  (f4)

Example

If  $t_{CK\_PSC} = 125$  ns (8 MHz), deadtime possible values are:

$DTG[7:0] = 0 \times 0$  to  $0 \times 7F$  from 0 to 15875 ns in 125 ns steps (refer to f1)

$DTG[7:0] = 0 \times 80$  to  $0 \times BF$  from 16  $\mu$ s to 31750 ns in 250 ns steps (refer to f2)

$DTG[7:0] = 0 \times C0$  to  $0 \times DF$  from 32  $\mu$ s to 63  $\mu$ s in 1  $\mu$ s steps (refer to f3)

$DTG[7:0] = 0 \times E0$  to  $0 \times FF$  from 64  $\mu$ s to 126  $\mu$ s in 2  $\mu$ s steps (refer to f4)

*Note: This bitfield can not be modified while LOCK level 1, 2, or 3 have been programmed (LOCK bits in the TIM1\_BKR register).*

### 17.7.32 Output idle state register (TIM1\_OISR)

Address offset: 0x1F

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
r	rw	rw	rw	rw	rw	rw	rw

Bit 7 Reserved, forced by hardware to 0

Bit 6 **OIS4**: Output idle state 4 (OC4 output)

Refer to OIS1 bit

Bit 5 **OIS3N**: Output idle state 3 (OC3N output)

Refer to OIS1N bit

Bit 4 **OIS3**: Output idle state 3 (OC3 output)

Refer to OIS1 bit

Bit 3 **OIS2N**: Output idle state 2 (OC2N output)

Refer to OIS1N bit

Bit 2 **OIS2**: Output idle state 2 (OC2 output)

Refer to OIS1 bit

Bit 1 **OIS1N**: Output idle state 1 (OC1N output).

0: OC1N = 0 after a deadtime when MOE = 0

1: OC1N = 1 after a deadtime when MOE = 0

*Note:* This bit can no longer be modified while LOCK level 1, 2 or 3 have been programmed (LOCK bits in the TIM1\_BKR register).

Bit 0 **OIS1**: Output idle state 1 (OC1 output).

0: OC1=0 (after a deadtime if OC1N is implemented) when MOE=0

1: OC1=1 (after a deadtime if OC1N is implemented) when MOE=0

*Note:* This bit can no longer be modified while LOCK level 1, 2 or 3 have been programmed (LOCK bits in the TIM1\_BKR register).

### 17.7.33 TIM1 register map and reset values

Table 39. TIM1 register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	TIM1_CR1 Reset value	ARPE 0	CMS1 0	CMS0 0	DIR 0	OPM 0	URS 0	UDIS 0	CEN 0
0x01	TIM1_CR2 Reset value	- 0	MMS2 0	MMS1 0	MMS0 0	- 0	COMS 0	- 0	CCPC 0
0x02	TIM1_SMCR Reset value	MSM 0	TS2 0	TS1 0	TS0 0	- 0	SMS2 0	SMS1 0	SMS0 0
0x03	TIM1_ETR Reset value	ETP 0	ECE 0	ETPS1 0	ETPS0 0	EFT3 0	EFT2 0	EFT1 0	EFT0 0
0x04	TIM1_IER Reset value	BIE 0	TIE 0	COMIE 0	CC4IE 0	CC3IE 0	CC2IE 0	CC1IE 0	UIE 0
0x05	TIM1_SR1 Reset value	BIF 0	TIF 0	COMIF 0	CC4IF 0	CC3IF 0	CC2IF 0	CC1IF 0	UIF 0
0x06	TIM1_SR2 Reset value	- 0	- 0	- 0	CC4OF 0	CC3OF 0	CC2OF 0	CC1OF 0	- 0
0x07	TIM1_EGR Reset value	BG 0	TG 0	COMG 0	CC4G 0	CC3G 0	CC2G 0	CC1G 0	UG 0
0x08	TIM1_CCMR1 (output mode) Reset value	OC1CE 0	OC1M2 0	OC1M1 0	OC1M0 0	OC1PE 0	OC1FE 0	CC1S1 0	CC1S0 0
	TIM1_CCMR1 (input mode) Reset value	IC1F3 0	IC1F2 0	IC1F1 0	IC1F0 0	IC1PSC 1 0	IC1PSC 0 0	CC1S1 0	CC1S0 0
0x09	TIM1_CCMR2 (output mode) Reset value	OC2CE 0	OC2M2 0	OC2M1 0	OC2M0 0	OC2PE 0	OC2FE 0	CC2S1 0	CC2S0 0
	TIM1_CCMR2 (input mode) Reset value	IC2F3 0	IC2F2 0	IC2F1 0	IC2F0 0	IC2PSC 1 0	IC2PSC 0 0	CC2S1 0	CC2S0 0
0x0A	TIM1_CCMR3 (output mode) Reset value	OC3CE 0	OC3M2 0	OC3M1 0	OC3M0 0	OC3PE 0	OC3FE 0	CC3S1 0	CC3S0 0
	TIM1_CCMR3 (input mode) Reset value	IC3F3 0	IC3F2 0	IC3F1 0	IC3F0 0	IC3PSC 1 0	IC3PSC 0 0	CC3S1 0	CC3S0 0

Table 39. TIM1 register map (continued)

Address offset	Register name	7	6	5	4	3	2	1	0
0x0B	TIM1_CCMR4 (output mode) Reset value	OC4CE 0	OC4M2 0	OC4M1 0	OC4M0 0	OC4PE 0	OC4FE 0	CC4S1 0	CC4S0 0
	TIM1_CCMR4 (input mode) Reset value	IC4F3 0	IC4F2 0	IC4F1 0	IC4F0 0	IC4PSC 1 0	IC4PSC 0 0	CC4S1 0	CC4S0 0
0x0C	TIM1_CCER1 Reset value	CC2NP 0	CC2NE 0	CC2P 0	CC2E 0	CC1NP 0	CC1NE 0	CC1P 0	CC1E 0
0x0D	TIM1_CCER2 Reset value	- 0	- 0	CC4P 0	CC4E 0	CC3NP 0	CC3NE 0	CC3P 0	CC3E 0
0x0E	TIM1_CNTRH Reset value	CNT15 0	CNT14 0	CNT13 0	CNT12 0	CNT11 0	CNT10 0	CNT9 0	CNT8 0
0x0F	TIM1_CNTRL Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x10	TIM1_PSCRH Reset value	PSC15 0	PSC14 0	PSC13 0	PSC12 0	PSC11 0	PSC10 0	PSC9 0	PSC8 0
0x11	TIM1_PSCRL Reset value	PSC7 0	PSC6 0	PSC5 0	PSC4 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0
0x12	TIM1_ARRH Reset value	ARR15 1	ARR14 1	ARR13 1	ARR12 1	ARR11 1	ARR10 1	ARR9 1	ARR8 1
0x13	TIM1_ARRL Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1
0x14	TIM1_RCR Reset value	REP7 0	REP6 0	REP5 0	REP4 0	REP3 0	REP2 0	REP1 0	REP0 0
0x15	TIM1_CCR1H Reset value	CCR115 0	CCR114 0	CCR113 0	CCR112 0	CCR111 0	CCR110 0	CCR19 0	CCR18 0
0x16	TIM1_CCR1L Reset value	CCR17 0	CCR16 0	CCR15 0	CCR14 0	CCR13 0	CCR12 0	CCR11 0	CCR10 0
0x17	TIM1_CCR2H Reset value	CCR215 0	CCR214 0	CCR213 0	CCR212 0	CCR211 0	CCR210 0	CCR29 0	CCR28 0
0x18	TIM1_CCR2L Reset value	CCR27 0	CCR26 0	CCR25 0	CCR24 0	CCR23 0	CCR22 0	CCR21 0	CCR20 0
0x19	TIM1_CCR3H Reset value	CCR315 0	CCR314 0	CCR313 0	CCR312 0	CCR311 0	CCR310 0	CCR39 0	CCR38 0
0x1A	TIM1_CCR3L Reset value	CCR37 0	CCR36 0	CCR35 0	CCR34 0	CCR33 0	CCR32 0	CCR31 0	CCR30 0
0x1B	TIM1_CCR4H Reset value	CCR415 0	CCR414 0	CCR413 0	CCR412 0	CCR411 0	CCR410 0	CCR49 0	CCR48 0
0x1C	TIM1_CCR4L Reset value	CCR47 0	CCR46 0	CCR45 0	CCR44 0	CCR43 0	CCR42 0	CCR41 0	CCR40 0

**Table 39. TIM1 register map (continued)**

<b>Address offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x1D	TIM1_BKR Reset value	MOE 0	AOE 0	BKP 0	BKE 0	OSSR 0	OSSI 0	LOCK 0	LOCK 0
0x1E	TIM1_DTR Reset value	DTG7 0	DTG6 0	DTG5 0	DTG4 0	DTG3 0	DTG2 0	DTG1 0	DTG0 0
0x1F	TIM1_OISR Reset value	- 0	OIS4 0	OIS3N 0	OIS3 0	OIS2N 0	OIS2 0	OIS1N 0	OIS1 0

## 18 16-bit general purpose timers (TIM2, TIM3, TIM5)

### 18.1 TIM2, TIM3 and TIM5 introduction

This chapter describes TIM2, TIM3 and TIM5 which are identical timers, with the exception that TIM2 has three channels and TIM3 has two channels. TIM5 is also described below. It is identical to TIM2 except that it has two additional registers to support timer synchronization and chaining.

Each timer consists of a 16-bit up-counting auto-reload counter driven by a programmable prescaler.

It 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)
- Synchronization with other timers or external signals (external clock, reset, trigger and enable) (in devices with TIM5).

The timer clock can be sourced from internal clocks.

Only the main features of the general purpose timers are given in this chapter.

Refer to the corresponding paragraphs of [Section 17: 16-bit advanced control timer \(TIM1\) on page 138](#) for more details on each feature.

### 18.2 TIM2/TIM3 main features

TIM2/TIM3 features include:

- 16-bit up counting auto-reload counter.
- 4-bit programmable prescaler allowing the counter clock frequency to be divided “on the fly” by any power of 2 from 1 to 32768.
- 3 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (edge-aligned mode)
- Interrupt request generation on the following events:
  - Update: counter overflow, counter initialization (by software)
  - Input capture
  - Output compare

## 18.3 TIM5 main features

TIM5 features include:

- 16-bit up counting auto-reload counter.
- 4-bit programmable prescaler allowing the counter clock frequency to be divided “on the fly” by any power of 2 from 1 to 32768.
- 3 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (edge-aligned mode)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers (See [Section 17.4.6 on page 159](#))
- External trigger input TIM1\_ETR (shared with TIM1)
- Interrupt generation on the following events:
  - Update: counter overflow, counter initialization (by software)
  - Input capture
  - Output compare

## 18.4 TIM2/TIM3/TIM5 functional description

Figure 82. TIM2/TIM3 block diagram

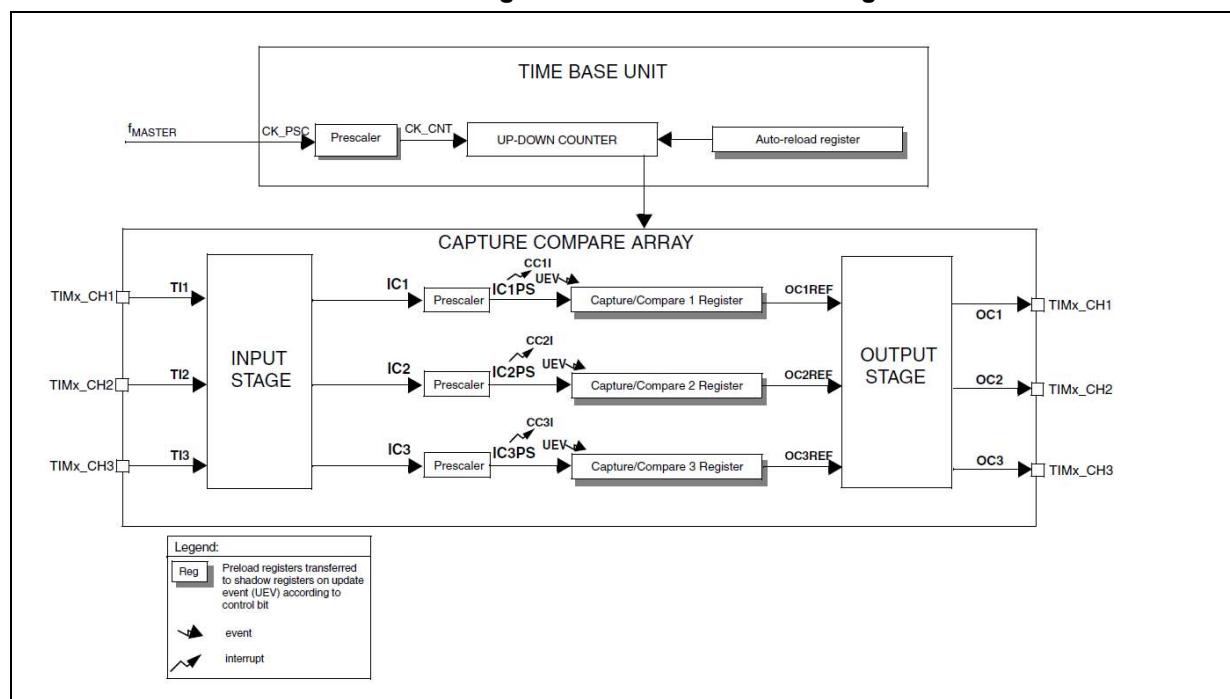
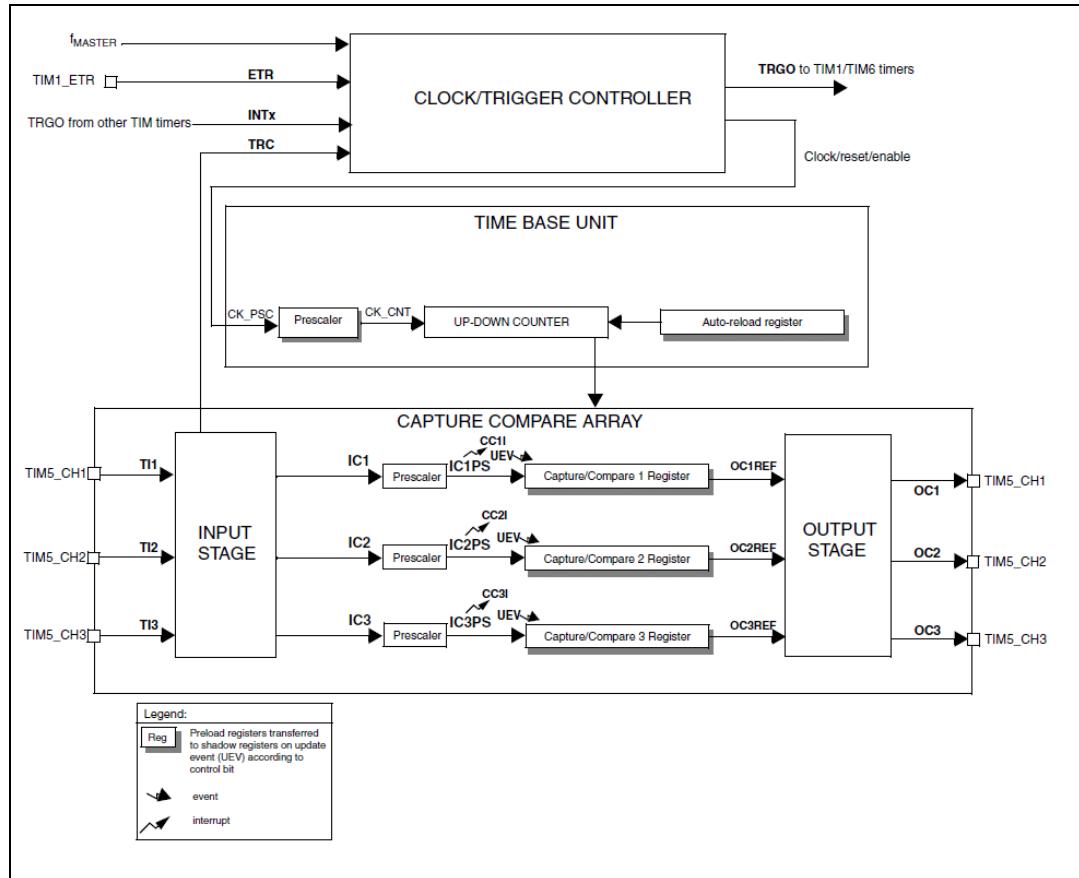


Figure 83. TIM5 block diagram



#### 18.4.1 Time base unit

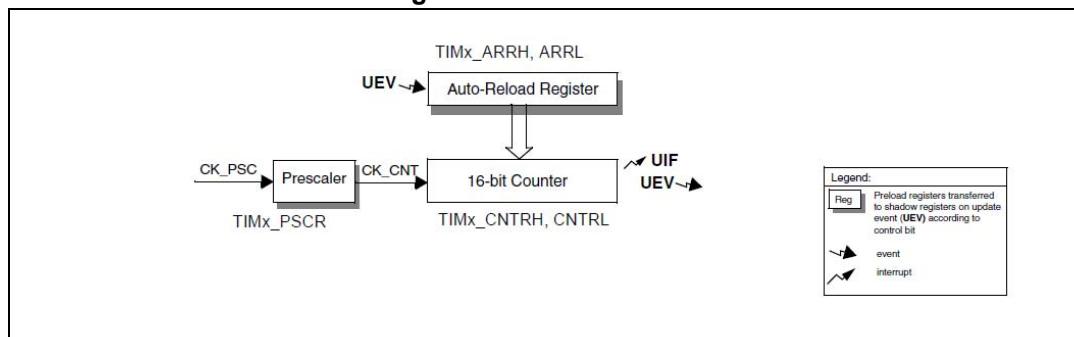
The timer has a time base unit that includes:

- 16-bit up counter
- 16-bit auto-reload register
- 4-bit programmable prescaler

There is no repetition counter.

The clock source for 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.

Figure 84. Time base unit



For more details refer to [Section 17.3: TIM1 time base unit on page 141](#).

### Prescaler

The prescaler implementation is as follows:

- The prescaler is based on a 16-bit counter controlled through a 4-bit register (in the TIMx\_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.

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

Read operations to the TIMx\_PSCR registers access the preload registers, so no special care needs to be taken to read them.

### Counter operation

Refer to [Section 17.3.4: Up-counting mode on page 143](#).

## 18.4.2 Clock/trigger controller

A clock/trigger controller and the associated TIMx\_CR2 and TIMx\_SMCR registers are not implemented in TIM2/TIM3, only in TIM5. Refer to [Section 17.4: TIM1 clock/trigger controller on page 151](#)

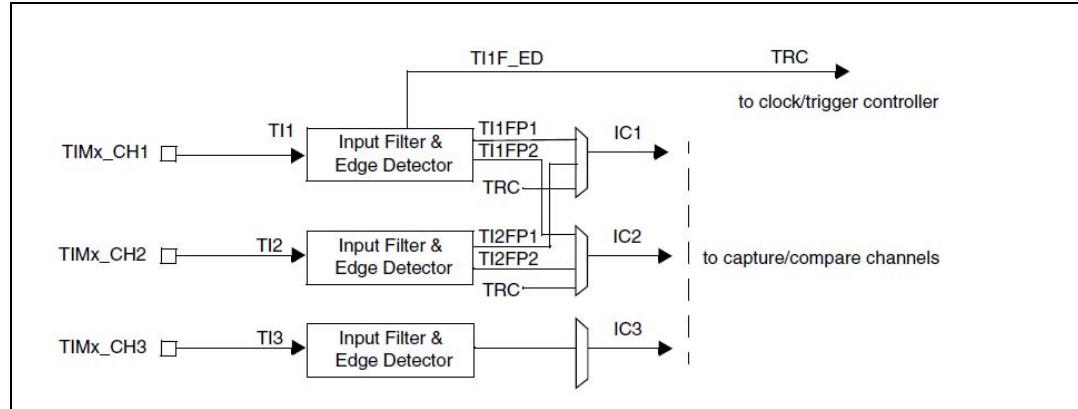
### 18.4.3 Capture/compare channels

#### Input stage

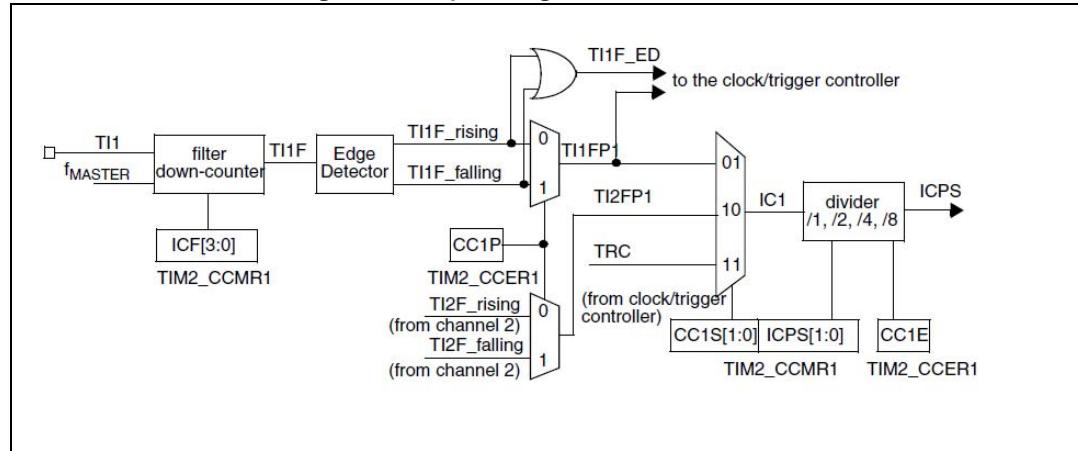
Refer to [Section 17.5: TIM1 capture/compare channels on page 166](#).

There are two input channels, as shown in [Figure 85: Input stage block diagram](#).

**Figure 85. Input stage block diagram**



**Figure 86. Input stage of TIM 2 channel 1**



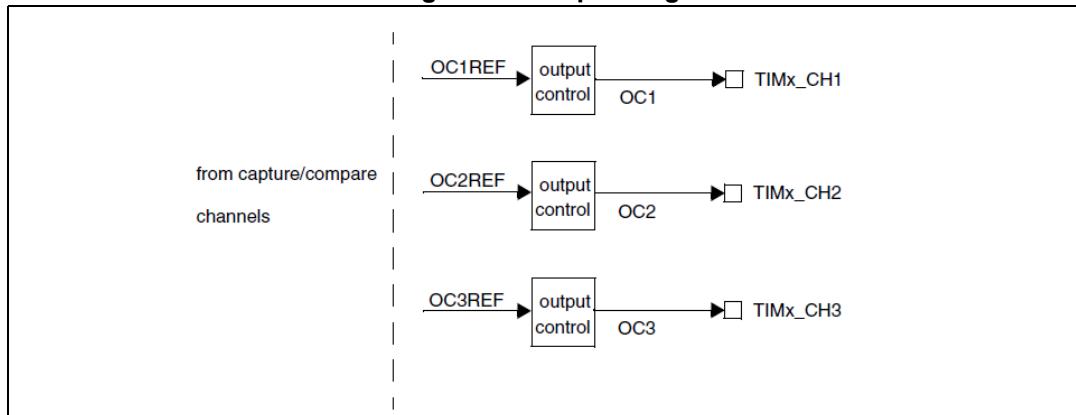
## Output stage

Refer to [Section 17.5.4: Output stage on page 171](#), [Section 17.5.5: Forced output mode on page 172](#), [Section 17.5.7: PWM mode on page 174](#).

**Note:** As the clock/trigger controller and the associated `TIMx_CR2` and `TIMx_SMCR` registers are not implemented in TIM2/TIM3, the one-pulse mode (described in [Section 17.5.7: PWM mode](#)) is not available in TIM2/TIM3.

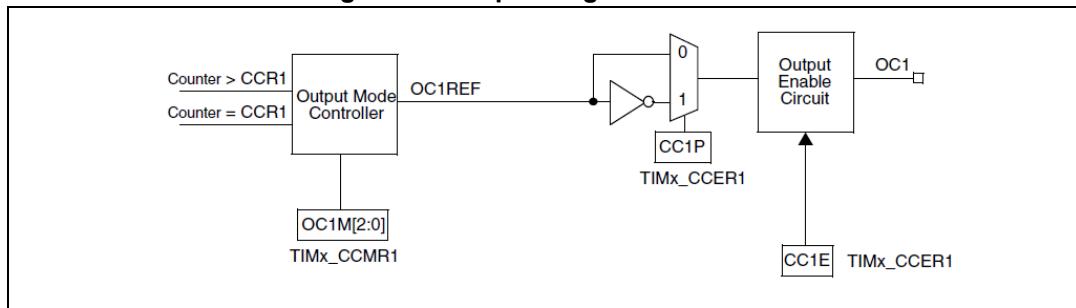
As shown in [Figure 87](#). TIMx outputs have no deadtime or complementary outputs.

**Figure 87. Output stage**



The output stage generates an intermediate waveform which is then used for reference: OC<sub>x</sub>REF (active high). Polarity acts at the end of the chain (see [Figure 88](#)).

**Figure 88. Output stage of channel 1**



## 18.5 TIM2/TIM3/TIM5 interrupts

The timers have 4 interrupt request sources:

- Capture/compare 3 interrupt
- Capture/compare 2 interrupt
- Capture/compare 1 interrupt
- Update interrupt
- Trigger interrupt (TIM5 only)

To use the interrupt features, for each interrupt channel used, set the desired CC3IE and/or CC2IE and/or CC1IE 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.

## 18.6 TIM2/TIM3/TIM5 registers

### 18.6.1 Control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
ARPE	Reserved		OPM	URS	UDIS	CEN	
rw	r		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 6:4 Reserved

Bit 3 **OPM**: One-pulse mode <sup>(1)</sup>

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 by the UDIS bit, the UIF bit is set and an update interrupt request is sent when one of the following events occurs:

- Registers are updated (counter overflow/underflow)
- UG bit is set by software
- Update event is generated through the clock/trigger controller

1: When enabled by the UDIS bit, the UIF bit is set and an update interrupt request is sent only when registers are updated (counter overflow/underflow).

Bit 1 **UDIS**: Update disable

0: A UEV 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: A UEV is not generated, shadow registers keep their value (ARR, PSC, CCR<sub>i</sub>). 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

1. One-pulse mode is not available on TIM2/TIM3 but the OPM bit can be used for other purposes (for example, to stop the counter properly).

## 18.6.2 Control register 2 (TIM5\_CR2)

*Address offset: 0x01*

*Reset value: 0x00*

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

*Note:* This register is only available in TIM5, see [Table 42 on page 250](#).

Bit 7 Reserved, must be kept cleared

Bits 6:4 **MMS[2:0]**: Master mode selection

These bits select the information to be sent in master mode to TIM1 and TIM2 for synchronization (TRGO). The combination is as follows:

000: Reset - the UG bit from the TIM5\_EGR register is used as a trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in trigger reset mode), the signal on TRGO is delayed compared to the actual reset.

001: Enable - the counter enable signal is used as a 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 the 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 TIM5\_SMCR register).

010: Update - The update event is selected as a trigger output (TRGO)

011: Reserved

100: Reserved

101: Reserved

111: Reserved

Bits 3:0 Reserved, must be kept cleared

### 18.6.3 Slave mode control register (TIM5\_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	rw	r	rw	rw	rw

*Note:* This register is only available in TIM5, see [Table 42 on page 250](#).

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 TIM6 TRGO (\*)

001: Internal trigger ITR1 connected to TIM6 TRGO (\*)

010: Reserved

011: Reserved

100: Reserved

101: Reserved

110: Reserved

111: Reserved

(\*) 8K low-density devices unless otherwise specified.

*Note:* These bits must only be changed 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, 010 and 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 the 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

### 18.6.4 Interrupt enable register (TIMx\_IER)

Address offset: 0x01 or 0x03 (TIM2), 0x01 (TIM3), 0x03 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TIE	Reserved		CC3IE	CC2IE	CC1IE	UIE
r	rw	r		rw	rw	rw	rw

Bits 7 Reserved

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled
- 1: Trigger interrupt enabled

*Note:* In TIM2/TIM3 this bit is reserved.

Bits 5:4 Reserved, must be kept cleared

Bit 3 **CC3IE**: Capture/compare 3 interrupt enable

- 0: CC3 interrupt disabled
- 1: CC3 interrupt enabled

Bit 2 **CC2IE**: Capture/compare 2 interrupt enable

- 0: CC2 interrupt disabled
- 1: CC2 interrupt enabled

Bit 1 **CC1IE**: Capture/compare 1 interrupt enable

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled
- 1: Update interrupt enabled

### 18.6.5 Status register 1 (TIMx\_SR1)

Address offset: 0x02 or 0x04 (TIM2), 0x02 (TIM3), 0x04 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TIF	Reserved		CC3IF	CC2IF	CC1IF	UIF
r	rc_w0	r		rc_w0	rc_w0	rc_w0	rc_w0

Bit 7 Reserved

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on a trigger event (active edge is detected on TRGI signal and both edges are detected when gated mode is selected). It is cleared by software.

0: No trigger event has occurred

1: Trigger interrupt pending

*Note: In TIM2/TIM3 this bit is reserved.*

Bits 5:4 Reserved, must be kept cleared

Bit 3 **CC3IF**: Capture/compare 3 interrupt flag

Refer to CC1IF description.

Bit 2 **CC2IF**: Capture/compare 2 interrupt flag

Refer to CC1IF description.

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

**If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value. 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 has 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 has 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.

### 18.6.6 Status register 2 (TIMx\_SR2)

Address offset: 0x03 or 0x05 (TIM2), 0x03 (TIM3), 0x05 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				CC3OF	CC2OF	CC1OF	Reserved
r				rc_w0	rc_w0	rc_w0	r

Bits 7:4 Reserved

Bit 3 **CC3OF**: Capture/compare 3 overcapture flag

Refer to CC1OF description

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, forced by hardware to 0

### 18.6.7 Event generation register (TIMx\_EGR)

Address offset: 0x04 or 0x06 (TIM2), 0x04 (TIM3), 0x06 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TG	Reserved		CC3G	CC2G	CC1G	UG
r	w	r		w	w	w	w

Bit 7 Reserved

Bit 6 **TG**: Trigger generation

This bit is set by software to generate an event. It is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIM5\_SR1 register. An interrupt is generated if enabled by the TIE bit

*Note: In TIM2/TIM3 this bit is reserved.*

Bits 5:4 Reserved

Bit 3 **CC3G**: Capture/compare 3 generation

Refer to CC1G description

Bit 2 **CC2G**: Capture/compare 2 generation

Refer to CC1G description

Bit 1 **CC1G**: Capture/compare 1 generation

This bit is set by software to generate an even. 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. In this case, the CC1IF flag is set, and the corresponding interrupt request is sent if enabled.
- If the CC1 channel configured in input mode. In this case, 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 is 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 also cleared.

### 18.6.8 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, OC<sub>i</sub> describes its function when the channel is configured in output and IC<sub>i</sub> describes its function when the channel is configured in input. Therefore, be aware that the same bit can have a different meaning for the input stage and for the output stage.

Address offset: 0x05 or 0x07 (TIM2), 0x05 (TIM3), 0x07 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

#### Channel configured in output

7	6	5	4	3	2	1	0
Reserved	OC1M[2:0]			OC1PE	Reserved	CC1S[1:0]	
r	rw	rw	rw	rw	r	rw	rw

Bit 7 Reserved

#### Bits 6:4 OC1M[2:0]: 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 matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1

100: Force inactive level - OC1REF is forced low

101: Force active level - OC1REF is forced high

110: PWM mode 1 - In up-counting, channel 1 is active as long as TIMx\_CNT < TIMx\_CCR1.

Otherwise, channel 1 is inactive. In down-counting, channel 1 is inactive (OC1REF = 0) as long as TIMx\_CNT > TIMx\_CCR1. Otherwise, channel 1 is active (OC1REF = 1).

111: PWM mode 2 - In up-counting, channel 1 is inactive as long as TIMx\_CNT < TIMx\_CCR1. Otherwise, channel 1 is active.

*Note: In PWM mode 1 or 2, the OC<sub>i</sub>REF 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 174](#) 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 into 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 Reserved****Bits 1:0 CC1S[1:0]:** Capture/compare 1 selection

This bitfield 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: CC1 channel is configured as input, IC1 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIM5\_SMCR register).

*Note: CC1S bits are writable only when the channel is off (CC1E = 0 and is updated in TIMx\_CCER1).*

### Channel configured in input

7	6	5	4	3	2	1	0
IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw

#### Bits 7:4 IC1F[3:0]: Input capture 1 filter

This bitfield 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[1:0]: Input capture 1 prescaler

This bitfield 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, the CC1E bit can be cleared and set again.*

#### Bits 1:0 CC1S[1:0]: Capture/compare 1 selection

This bitfield 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).*

### 18.6.9 Capture/compare mode register 2 (TIMx\_CCMR2)

**Note:** Refer to [Capture/compare mode register 1 \(TIMx\\_CCMR1\) on page 236](#) for details on using these bits.

Address offset: 0x06 or 0x08 (TIM2), 0x06 (TIM3), 0x08 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

#### Channel configured in output

7	6	5	4	3	2	1	0
Reserved	OC2M[2:0]			OC2PE	Reserved	CC2S[1:0]	
r	rw	rw	rw	rw	r	rw	

Bit 7 Reserved

Bits 6:4 **OC2M[2:0]**: Output compare 2 mode

Bit 3 **OC2PE**: Output compare 2 preload enable

Bit 2 Reserved

Bits 1:0 **CC2S[1:0]**: Capture/compare 2 selection

This bitfield 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: CC2 channel is configured as input, IC2 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIM5\_SMCR register).

**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	rw	rw	rw	rw	rw

Bits 7:4 **IC2F[3:0]**: Input capture 2 filter

Bits 3:2 **IC2PCS[1:0]**: Input capture 2 prescaler

Bits 1:0 **CC2S[1:0]**: Capture/compare 2 selection

This bitfield 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).

### 18.6.10 Capture/compare mode register 3 (TIMx\_CCMR3)

Refer to [Capture/compare mode register 1 \(TIM1\\_CCMR1\) on page 198](#) for details on using these bits.

Address offset: 0x07 or 0x09 (TIM2), 0x09 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

### Channel configured in output

7	6	5	4	3	2	1	0
Reserved	OC3M[2:0]				OC3PE	Reserved	CC3S[1:0]
r	rw	rw	rw	rw	rw	r	rw

*Note:* This register is not available in TIM3.

Bit 7 Reserved

Bits 6:4 **OC3M[2:0]**: Output compare 3 mode

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 Reserved

Bits 1:0 **CC3S[1:0]**: Capture/compare 3 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3FP3

10: Reserved

11: Reserved

*Note:* CC3S bits are writable only when the channel is off (CC3E = 0 in TIMx\_CCER2).

### Channel configured in input

7	6	5	4	3	2	1	0
IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw

Note: This register is not available in TIM3.

Bits 7:4 **IC3F[3:0]** Input capture 3 filter

Bits 3:2 **IC3PSC(1:0)**: Input capture 3 prescaler

Bits 1:0 **CC3S[1:0]**: Capture/compare 3 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3FP3

10: Reserved

11: Reserved

Note: CC3S bits are writable only when the channel is off (CC3E = 0 in TIMx\_CCER2).

### 18.6.11 Capture/compare enable register 1 (TIMx\_CCER1)

Address offset: 0x08 or 0x0A (TIM2), 0x07 (TIM3), 0x0A (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	CC2P	CC2E	Reserved	Reserved	CC1P	CC1E	
r	rw	rw	r	r	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 64):**

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 be made in the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled

1: Capture enabled

### 18.6.12 Capture/compare enable register 2 (TIMx\_CCER2)

Address offset: 0x09 or 0x0B (TIM2), 0x0B (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						CC3P	CC3E
r						rw	rw

Note: This register is not available in TIM3.

Bits 7:2 Reserved

Bit 1 **CC3P**: Capture/compare 3 output polarity

Refer to CC1P description.

Bit 0 **CC3E**: Capture/compare 3 output enable

Refer to CC1E description.

### 18.6.13 Counter high (TIMx\_CNTRH)

Address offset: 0x0A or 0x0C (TIM2), 0x08 (TIM3), 0x0C (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CNT[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CNT[15:8]**: Counter value (MSB)

### 18.6.14 Counter low (TIMx\_CNTRL)

Address offset: 0x0B or 0x0D (TIM2), 0x09 (TIM3), 0x0D (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

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

Bits 7:0 CNT[7:0]: Counter value (LSB)

### 18.6.15 Prescaler register (TIMx\_PSCR)

Address offset: 0x0C or 0x0E (TIM2), 0x0A (TIM3), 0x0E (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

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

Bits 7:3 Reserved

Bits 2:0 **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[7:4] are forced to 0 by hardware.

PSCR contains the value which is loaded in the active prescaler register at each update event (including when the counter is cleared through the UG bit of the TIMx\_EGR register).

This means that a UEV must be generated so that a new prescaler value can be taken into account.

### 18.6.16 Auto-reload register high (TIMx\_ARRH)

Address offset: 00x0D or 0x0F (TIM2), 0x0B (TIM3), 0x0F (TIM5); for TIM2 address see [Section](#)

Reset value: 0xFF

7	6	5	4	3	2	1	0
ARR[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **ARR[15:8]**: Auto-reload value (MSB)

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 17.3: TIM1 time base unit on page 141](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is 0.

### 18.6.17 Auto-reload register low (TIMx\_ARRL)

Address offset: 0x0E or 0x10 (TIM2), 0x0C (TIM3), 0x10 (TIM5); for TIM2 address see [Section](#)

Reset value: 0xFF

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

Bits 7:0 **ARR[7:0]**: Auto-reload value (LSB)

### 18.6.18 Capture/compare register 1 high (TIMx\_CCR1H)

Address offset: 0x0F or 0x11 (TIM2), 0x0D (TIM3), 0x11 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR1[15:8]							
rw	rw	rw	rw	rw	rw	rw	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):**

The value of CCR1 is loaded permanently into the actual capture/compare 1 register if the preload feature is not enabled (OC1PE bit in TIMx\_CCMR1). Otherwise, the preload value is copied in the active capture/compare 1 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIMx\_CNT, and signalled on the OC1 output.

**If the CC1 channel is configured as input (CC1S bits in TIMx\_CCMR1 register):**

The value of CCR1 is the counter value transferred by the last input capture 1 event (IC1). In this case, these bits are read only.

### 18.6.19 Capture/compare register 1 low (TIMx\_CCR1L)

Address offset: 00x10 or 0x12 (TIM2), 0x0E (TIM3), 0x12 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR1[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR1[7:0]**: Capture/compare 1 value (LSB)

### 18.6.20 Capture/compare register 2 high (TIMx\_CCR2H)

Address offset: 00x11 or 0x13 (TIM2), 0x0F (TIM3), 0x13 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR2[15:8]							
rw	rw	rw	rw	rw	rw	rw	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):**

The value of CCR2 is loaded permanently into the actual capture/compare 2 register if the preload feature is not enabled (OC2PE bit in TIMx\_CCMR2). Otherwise, the preload value is copied in the active capture/compare 2 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIMx\_CNT, and signalled on the OC2 output.

**If the CC2 channel is configured as input (CC2S bits in TIMx\_CCMR2 register):**

The value of CCR2 is the counter value transferred by the last input capture 2 event (IC2).

### 18.6.21 Capture/compare register 2 low (TIMx\_CCR2L)

Address offset: 00x12 or 0x14 (TIM2), 0x10 (TIM3), 0x14 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **CCR2[7:0]**: Capture/compare value (LSB)

### 18.6.22 Capture/compare register 3 high (TIMx\_CCR3H)

Address offset: 00x13 or 0x15 (TIM2), 0x15 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR3[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Note: *This register is not available in TIM3.*

Bits 7:0 CCR3[15:8]: Capture/compare value (MSB)

**If the CC3 channel is configured as output (CC3S bits in TIMx\_CCMR3 register):**

The value of CCR3 is loaded permanently into the actual capture/compare 3 register if the preload feature is not enabled (OC3PE bit in TIMx\_CCMR3). Otherwise, the preload value is copied in the active capture/compare 3 register when a UEV occurs. The active capture/compare register contains the value which is compared to the counter register, TIMx\_CNT, and signalled on the OC3 output.

**If the CC3 channel is configured as input (CC3S bits in TIMx\_CCMR3 register):**

The value of CCR3 is the counter value transferred by the last input capture 3 event (IC3).

### 18.6.23 Capture/compare register 3 low (TIMx\_CCR3L)

Address offset: 00x14 or 0x16 (TIM2), 0x16 (TIM5); for TIM2 address see [Section](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
CCR3[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

Note: *This register is not available in TIM3.*

Bits 7:0 CCR3[7:0]: Capture/compare value (LSB)

### TIM2/TIM3/TIM5 register map and reset values

In some STM8S and STM8AF devices, TIM2 register locations at offset 0x01 and 0x02 are reserved. In this case the TIM2\_IER and subsequent registers in the TIM2 block are offset by 2 more bytes. Refer to the datasheet for the product-specific register map.

Table 40. TIM2 register map

Address offset (product dependent)		Register name	7	6	5	4	3	2	1	0
0x00	0x00	TIM2_CR1 Reset value	ARPE 0	- 0	- 0	- 0	OPM 0	URS 0	UDIS 0	CEN 0
-	0x01	Reserved								
-	0x02	Reserved								

Table 40. TIM2 register map (continued)

Address offset (product dependent)		Register name	7	6	5	4	3	2	1	0
0x01	0x03	TIM2_IER Reset value	-0	-0	-0	-0	CC3IE 0	CC2IE 0	CC1IE 0	UIE 0
0x02	0x04	TIM2_SR1 Reset value	-0	-0	-0	-0	CC3IF 0	CC2IF 0	CC1IF 0	UIF 0
0x03	0x05	TIM2_SR2 Reset value	-0	-0	-0	-0	CC3OF 0	CC2OF 0	CC1OF 0	-0
0x04	0x06	TIM2_EGR Reset value	-0	-0	-0	-0	CC3G 0	CC2G 0	CC1G 0	UG 0
0x05	0x07	TIM2_CCMR 1 (output mode) Reset value	-0	OC1M2 0	OC1M1 0	OC1M0 0	OC1PE 0	-0	CC1S1 0	CC1S0 0
		TIM2_CCMR 1 (input mode) Reset value	IC1F3 0	IC1F2 0	IC1F1 0	IC1F0 0	IC1PSC 1 0	IC1PSC 0 0	CC1S1 0	CC1S0 0
0x06	0x08	TIM2_ CCMR2 (output mode) Reset value	-0	OC2M2 0	OC2M1 0	OC2M0 0	OC2PE 0	-0	CC2S1 0	CC2S0 0
		TIM2_CCMR 2 (input mode) Reset value	IC2F3 0	IC2F2 0	IC2F1 0	IC2F0 0	IC2PSC 1 0	IC2PSC 0 0	CC2S1 0	CC2S0 0
0x07	0x09	TIM2_CCMR 3 (output mode) Reset value	-0	OC3M2 0	OC3M1 0	OC3M0 0	OC3PE 0	-0	CC3S1 0	CC3S0 0
		TIM2_CCMR 3 (input mode) Reset value	IC3F3 0	IC3F2 0	IC3F1 0	IC3F0 0	IC3PSC 1 0	IC3PSC 0 0	CC3S1 0	CC3S0 0
0x08	0x0A	TIM2_CCER 1 Reset value	-0	-0	CC2P 0	CC2E 0	-0	-0	CC1P 0	CC1E 0
0x09	0x0B	TIM2_CCER 2 Reset value	-0	-0	-0	-0	-0	-0	CC3P 0	CC3E 0
0x0A	0x0C	TIM2_CNTR H Reset value	CNT15 0	CNT14 0	CNT13 0	CNT12 0	CNT11 0	CNT10 0	CNT9 0	CNT8 0

Table 40. TIM2 register map (continued)

Address offset (product dependent)		Register name	7	6	5	4	3	2	1	0
0x0B	0x0D	TIM2_CNTRL Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x0C	0x0E	TIM2_PSCR Reset value	- 0	- 0	- 0	- 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0
0x0D	0x0F	TIM2_ARRH Reset value	ARR15 1	ARR14 1	ARR13 1	ARR12 1	ARR11 1	ARR10 1	ARR9 1	ARR8 1
0x0E	0x10	TIM2_ARRL Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1
0x0F	0x11	TIM2_CCR1H Reset value	CCR115 0	CCR114 0	CCR113 0	CCR112 0	CCR111 0	CCR110 0	CCR19 0	CCR18 0
0x10	0x12	TIM2_CCR1L Reset value	CCR17 0	CCR16 0	CCR15 0	CCR14 0	CCR13 0	CCR12 0	CCR11 0	CCR10 0
0x11	0x13	TIM2_CCR2H Reset value	CCR215 0	CCR214 0	CCR213 0	CCR212 0	CCR211 0	CCR210 0	CCR29 0	CCR28 0
0x12	0x14	TIM2_CCR2L Reset value	CCR27 0	CCR26 0	CCR25 0	CCR24 0	CCR23 0	CCR22 0	CCR21 0	CCR20 0
0x13	0x15	TIM2_CCR3H Reset value	CCR315 0	CCR314 0	CCR313 0	CCR312 0	CCR311 0	CCR310 0	CCR39 0	CCR38 0
0x14	0x16	TIM2_CCR3L Reset value	CCR37 0	CCR36 0	CCR35 0	CCR34 0	CCR33 0	CCR32 0	CCR31 0	CCR30 0

Table 41. TIM3 register map

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	TIM3_CR1 Reset value	ARPE 0	- 0	- 0	- 0	OPM 0	URS 0	UDIS 0	CEN 0
0x01	TIM3_IER Reset value	- 0	- 0	- 0	- 0	- 0	CC2IE 0	CC1IE 0	UIE 0
0x02	TIM3_SR1 Reset value	- 0	- 0	- 0	- 0	- 0	CC2IF 0	CC1IF 0	UIF 0
0x03	TIM3_SR2 Reset value	- 0	- 0	- 0	- 0	- 0	CC2OF 0	CC1OF 0	- 0
0x04	TIM3_EGR Reset value	- 0	- 0	- 0	- 0	- 0	CC2G 0	CC1G 0	UG 0

Table 41. TIM3 register map (continued)

Address offset	Register name	7	6	5	4	3	2	1	0
0x05	TIM3_CCMR1 (output mode) Reset value	- 0	OC1M2 0	OC1M1 0	OC1M0 0	OC1PE 0	- 0	CC1S1 0	CC1S0 0
	TIM3_CCMR1 (input mode) Reset value	IC1F3 0	IC1F2 0	IC1F1 0	IC1F0 0	IC1PSC 1 0	IC1PSC 0 0	CC1S1 0	CC1S0 0
0x06	TIM3_CCMR2 (output mode) Reset value	- 0	OC2M2 0	OC2M1 0	OC2M0 0	OC2PE 0	- 0	CC2S1 0	CC2S0 0
	TIM3_CCMR2 (input mode) Reset value	IC2F3 0	IC2F2 0	IC2F1 0	IC2F0 0	IC2PSC 1 0	IC2PSC 0 0	CC2S1 0	CC2S0 0
0x07	TIM3_CCER1 Reset value	- 0	- 0	CC2P 0	CC2E 0	- 0	- 0	CC1P 0	CC1E 0
0x08	TIM3_CNTRH Reset value	CNT15 0	CNT14 0	CNT13 0	CNT12 0	CNT11 0	CNT10 0	CNT9 0	CNT8 0
0x09	TIM3_CNTRL Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x0A	TIM3_PSCR Reset value	- 0	- 0	- 0	- 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0
0x0B	TIM3_ARRH Reset value	ARR15 1	ARR14 1	ARR13 1	ARR12 1	ARR11 1	ARR10 1	ARR9 1	ARR8 1
0x0C	TIM3_ARRL Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1
0x0D	TIM3_CCR1H Reset value	CCR115 0	CCR114 0	CCR113 0	CCR112 0	CCR111 0	CCR110 0	CCR19 0	CCR18 0
0x0E	TIM3_CCR1L Reset value	CCR17 0	CCR16 0	CCR15 0	CCR14 0	CCR13 0	CCR12 0	CCR11 0	CCR10 0
0x0F	TIM3_CCR2H Reset value	CCR215 0	CCR214 0	CCR213 0	CCR212 0	CCR211 0	CCR210 0	CCR29 0	CCR28 0
0x10h	TIM3_CCR2L Reset value	CCR27 0	CCR26 0	CCR25 0	CCR24 0	CCR23 0	CCR22 0	CCR21 0	CCR20 0

Table 42. TIM5 register map

Address	Register name	7	6	5	4	3	2	1	0
0x00	TIM5_CR1 Reset value	ARPE 0	- 0	- 0	- 0	OPM 0	URS 0	UDIS 0	CEN 0
0x01	TIM5_CR2 Reset value	- 0	MMS2 0	MMS1 0	MMS0 0	- 0	COMS 0	- 0	CCPC 0

Table 42. TIM5 register map (continued)

Address	Register name	7	6	5	4	3	2	1	0
0x02	TIM5_SMCR Reset value	MSM 0	TS2 0	TS1 0	TS0 0	- 0	SMS2 0	SMS1 0	SMS0 0
0x03	TIM5_IER Reset value	- 0	TIE 0	- 0	- 0	CC3IE 0	CC2IE 0	CC1IE 0	UIE 0
0x04	TIM5_SR1 Reset value	- 0	TIF 0	- 0	- 0	CC3IF 0	CC2IF 0	CC1IF 0	UIF 0
0x05	TIM5_SR2 Reset value	- 0	- 0	- 0	- 0	CC3OF 0	CC2OF 0	CC1OF 0	- 0
0x06	TIM5_EGR Reset value	- 0	TG 0	- 0	- 0	CC3G 0	CC2G 0	CC1G 0	UG 0
0x07	TIM5_CCMR1 (output mode) Reset value	- 0	OC1M2 0	OC1M1 0	OC1M0 0	OC1PE 0	- 0	CC1S1 0	CC1S0 0
	TIM5_CCMR1 (input mode) Reset value	IC1F3 0	IC1F2 0	IC1F1 0	IC1F0 0	IC1PSC 1 0	IC1PSC 0 0	CC1S1 0	CC1S0 0
0x08	TIM5_CCMR2 (output mode) Reset value	- 0	OC2M2 0	OC2M1 0	OC2M0 0	OC2PE 0	- 0	CC2S1 0	CC2S0 0
	TIM5_CCMR2 (input mode) Reset value	IC2F3 0	IC2F2 0	IC2F1 0	IC2F0 0	IC2PSC 1 0	IC2PSC 0 0	CC2S1 0	CC2S0 0
0x09	TIM5_CCMR3 (output mode) Reset value	- 0	OC3M2 0	OC3M1 0	OC3M0 0	OC3PE 0	- 0	CC3S1 0	CC3S0 0
	TIM5_CCMR3 (input mode) Reset value	IC3F3 0	IC3F2 0	IC3F1 0	IC3F0 0	IC3PSC 1 0	IC3PSC 0 0	CC3S1 0	CC3S0 0
0x0A	TIM5_CCER1 Reset value	- 0	- 0	CC2P 0	CC2E 0	- 0	- 0	CC1P 0	CC1E 0
0x0B	TIM5_CCER2 Reset value	- 0	- 0	- 0	- 0	- 0	- 0	CC3P 0	CC3E 0
0x0C	TIM5_CNTRH Reset value	CNT15 0	CNT14 0	CNT13 0	CNT12 0	CNT11 0	CNT10 0	CNT9 0	CNT8 0
0x0D	TIM5_CNTRL Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x0E	TIM5_PSCR Reset value	- 0	- 0	- 0	- 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0
0x0F	TIM5_ARRH Reset value	ARR15 1	ARR14 1	ARR13 1	ARR12 1	ARR11 1	ARR10 1	ARR9 1	ARR8 1

**Table 42. TIM5 register map (continued)**

<b>Address</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x10	TIM5_ARRL Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1
0x11	TIM5_CCR1H Reset value	CCR115 0	CCR114 0	CCR113 0	CCR112 0	CCR111 0	CCR110 0	CCR19 0	CCR18 0
0x12	TIM5_CCR1L Reset value	CCR17 0	CCR16 0	CCR15 0	CCR14 0	CCR13 0	CCR12 0	CCR11 0	CCR10 0
0x13	TIM5_CCR2H Reset value	CCR215 0	CCR214 0	CCR213 0	CCR212 0	CCR211 0	CCR210 0	CCR29 0	CCR28 0
0x14	TIM5_CCR2L Reset value	CCR27 0	CCR26 0	CCR25 0	CCR24 0	CCR23 0	CCR22 0	CCR21 0	CCR20 0
0x15	TIM5_CCR3H Reset value	CCR315 0	CCR314 0	CCR313 0	CCR312 0	CCR311 0	CCR310 0	CCR39 0	CCR38 0
0x16	TIM5_CCR3L Reset value	CCR37 0	CCR36 0	CCR35 0	CCR34 0	CCR33 0	CCR32 0	CCR31 0	CCR30 0

## 19 8-bit basic timer (TIM4, TIM6)

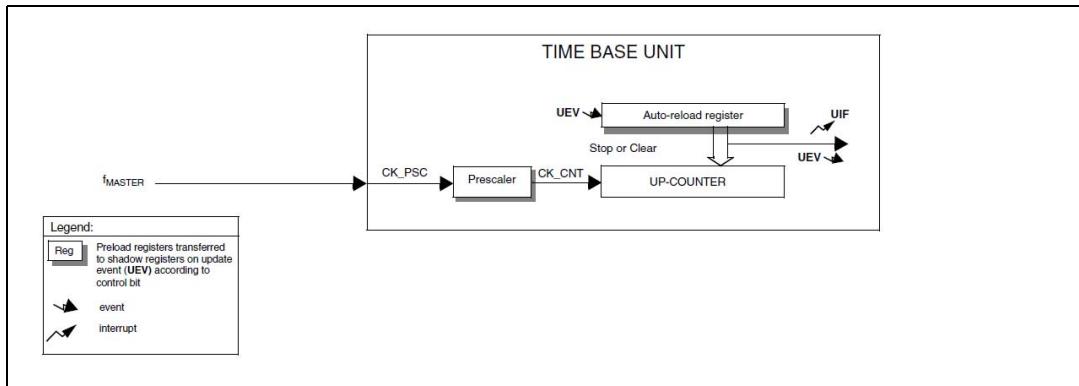
### 19.1 TIM4, TIM6 introduction

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.

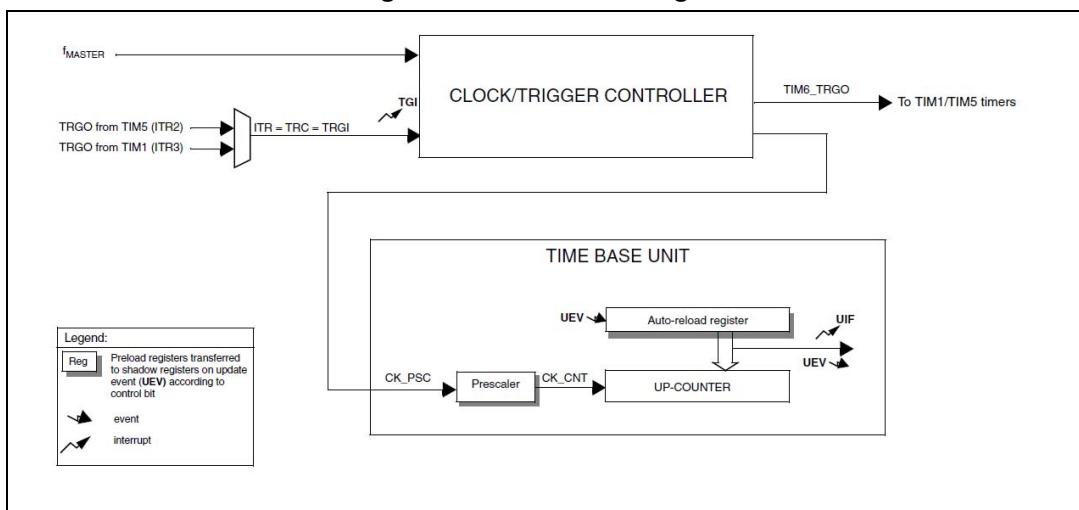
TIM6 is implemented with the clock/trigger controller for timer synchronization and chaining.

Refer to [Section 17.3 on page 141](#) for the general description of the timer features.

**Figure 89. TIM4 block diagram**



**Figure 90. TIM6 block diagram**



## 19.2 TIM4 main features

The main features include:

- 8-bit auto-reload up counter
- 3-bit programmable prescaler which allows dividing (also “on the fly”) the counter clock frequency by 1, 2, 4, 8, 16, 32, 64 and 128.
- Interrupt generation
  - On counter update: Counter overflow

## 19.3 TIM6 main features

The main features include:

- 8-bit auto-reload up counter
- 3-bit programmable prescaler which allows dividing (also “on the fly”) the counter clock frequency by 1, 2, 4, 8, 16, 32, 64 and 128.
- Synchronization circuit to control the timer with external signals and to interconnect several timers (See [Section 17.4.6 on page 159](#)).
- Interrupt generation
  - On counter update: Counter overflow
  - On trigger input

## 19.4 TIM4/TIM6 interrupts

The timer has 2 interrupt request sources:

- Update interrupt (overflow, counter initialization)
- Trigger input (TIM6 only)

## 19.5 TIM4/TIM6 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 prescaler is based on a 7-bit counter controlled through a 3-bit register (in the TIMx\_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 128.

The counter clock frequency is calculated as follows:

$$f_{CK\_CNT} = f_{CK\_PSC}/2^{(PSCR[2:0])}$$

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 TIMx\_PSCR registers access the preload registers, so no special care needs to be taken to read them.

## 19.6 TIM4/TIM6 registers

### 19.6.1 Control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
ARPE		Reserved		OPM	URS	UDIS	CEN

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, must be kept cleared

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: A UEV 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: A UEV 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

### 19.6.2 Control register 2 (TIM6\_CR2)

Address offset: 0x01

Reset value: 0x00

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

*Note:* This register is not available in TIM4.

Bit 7 Reserved, must be kept cleared

Bits 6:4 MMS[2:0]: Master mode selection

These bits are used to select the information to be sent in master mode to for synchronization (TRGO). The combination is as follows:

000: Reset - the UG bit from the TIM6\_EGR register is used as a trigger output (TRGO). If the reset is generated by the trigger input (clock/trigger mode controller configured in trigger reset mode), the signal on the TRGO is delayed compared to the actual reset.

001: Enable - the counter enable signal is used as a 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 the 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 TIM6\_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, must be kept cleared

### 19.6.3 Slave mode control register (TIM6\_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	rw	r	rw	rw	rw

*Note:* This register is not available in TIM4.

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: Internal trigger ITR1 connected to TIM1 TRGO (\*)

010: Reserved

011: Internal trigger ITR3 connected to TIM5 TRGO (\*)

100: Reserved

101: Reserved

110: Reserved

111: Reserved

(\*) 8K low-density devices unless otherwise specified.

*Note: These bits must only be changed 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, the prescaler is clocked directly by the internal clock.

001: Reserved

010: Reserved

011: Reserved

100: Trigger reset mode - The 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.

### 19.6.4 Interrupt enable register (TIMx\_IER)

Address offset: 0x01 or 0x03 (TIM4), 0x03 (TIM6); for TIM4 address see [Section 19.6.10](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TIE			Reserved			UIE
r	rw			r			rw

Bit 7 Reserved, must be kept cleared

Bit 6 **TIE**: Trigger interrupt enable

0: Trigger Interrupt disabled

1: Trigger Interrupt enabled

*Note: In TIM4 this bit is reserved.*

Bits 5:1 Reserved, must be kept cleared

Bit 0 **UIE**: Update interrupt enable

0: Update interrupt disabled

1: Update interrupt enabled

### 19.6.5 Status register 1 (TIMx\_SR)

Address offset: 0x02 or 0x04 (TIM4), 0x04 (TIM6); for TIM4 address see [Section 19.6.10](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TIF			Reserved			UIF
	rc_w0						rc_w0

Bit 7 Reserved, must be kept cleared

Bit 6 **TIF**: Trigger interrupt flag.

This flag is set by hardware on a trigger event (the active edge is detected on the TRGI signal, both edges are detected if gated mode is selected). It is cleared by software.

0: No trigger event has occurred

1: Trigger interrupt pending.

*Note: In TIM4 this bit is reserved.*

Bits 5:1 Reserved, must be kept cleared

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update has 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.

### 19.6.6 Event generation register (TIMx\_EGR)

Address offset: 0x03 or 0x05 (TIM4), 0x05 (TIM6); for TIM4 address see [Section 19.6.10](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TG W			Reserved			UG W

Bit 7 Reserved, must be kept cleared

Bit 6 **TG**: Trigger generation

This bit is set by software to generate an event. It is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIM6\_SR register. An interrupt is generated if enabled by the TIE bit

*Note: In TIM4 this bit is reserved.*

Bits 5:1 Reserved, must be kept cleared

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 also cleared.

### 19.6.7 Counter (TIMx\_CNTR)

Address offset: 0x04 or 0x06 (TIM4), 0x06 (TIM6); for TIM4 address see [Section 19.6.10](#)

Reset value: 0x00

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

Bits 7:0 **CNT[7:0]**: Counter value

### 19.6.8 Prescaler register (TIMx\_PSCR)

Address offset: 0x05 or 0x07 (TIM4), 0x07 (TIM6); for TIM4 address see [Section 19.6.10](#)

Reset value: 0x00

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

Bits 7:3 Reserved, must be kept cleared

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 contains the value which is loaded into the active prescaler register at each UEV (including when the counter is cleared through the UG bit of TIM4\_EGR).

Consequently, a UEV must be generated so that a new prescaler value can be taken into account.

### 19.6.9 Auto-reload register (TIMx\_ARR)

Address offset: 0x06 or 0x08 (TIM4), 0x08 (TIM6); for TIM4 address see [Section 19.6.10](#)

Reset value: 0xFF

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

Bits 7:0 **ARR[7:0]**: Auto-reload value

### 19.6.10 TIM4/TIM6 register map and reset values

In some STM8S and STM8AF devices, TIM4 register locations at offset 0x01 and 0x02 are reserved. In this case the TIM4\_IER and subsequent registers in the TIM4 block are offset by 2 more bytes. Refer to the datasheet for the product-specific register map.

Table 43. TIM4 register map

Address offset (product dependent)		Register name	7	6	5	4	3	2	1	0
0x00	0x00	TIM4_CR1 Reset value	ARPE 0	- 0	- 0	- 0	OPM 0	URS 0	UDIS 0	CEN 0
-	0x01	Reserved								
-	0x02	Reserved								
0x01	0x03	TIM4_IER Reset value	- 0	UIE 0						
0x02	0x04	TIM4_SR Reset value	- 0	UIF 0						
0x03	0x05	TIM4_EGR Reset value	- 0	UG 0						
0x04	0x06	TIM4_CNT Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x05	0x07	TIM4_PSCR Reset value	- 0	- 0	- 0	- 0	- 0	PSC2 0	PSC1 0	PSC0 0
0x06	0x08	TIM4_ARR Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1

**Table 44. TIM6 register map**

<b>Address offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x00	TIM6_CR1 Reset value	ARPE 0	- 0	- 0	- 0	OPM 0	URS 0	UDIS 0	CEN 0
0x01	TIM6_CR2 Reset value	- 0	MMS2 0	MMS1 0	MMS0 0	- 0	- 0	- 0	- 0
0x02	TIM6_SMCR Reset value	MSM 0	TS2 0	TS1 0	TS0 0	- 0	SMS2 0	SMS1 0	SMS0 0
0x03	TIM6_IER Reset value	- 0	TIE 0	- 0	- 0	- 0	- 0	- 0	UIE 0
0x04	TIM6_SR1 Reset value	- 0	TIF 0	- 0	- 0	- 0	- 0	- 0	UIF 0
0x05	TIM6_EGR Reset value	- 0	TG 0	- 0	- 0	- 0	- 0	- 0	UG 0
0x06	TIM6_CNTR Reset value	CNT7 0	CNT6 0	CNT5 0	CNT4 0	CNT3 0	CNT2 0	CNT1 0	CNT0 0
0x07	TIM6_PSCR Reset value	- 0	- 0	- 0	- 0	- 0	PSC2 0	PSC1 0	PSC0 0
0x08	TIM6_ARR Reset value	ARR7 1	ARR6 1	ARR5 1	ARR4 1	ARR3 1	ARR2 1	ARR1 1	ARR0 1

## 20 Serial peripheral interface (SPI)

### 20.1 SPI 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.

It may be used for a variety of purposes, including simplex synchronous transfers on 2 lines with a possible bidirectional data line or reliable communication using CRC checking.

### 20.2 SPI main features

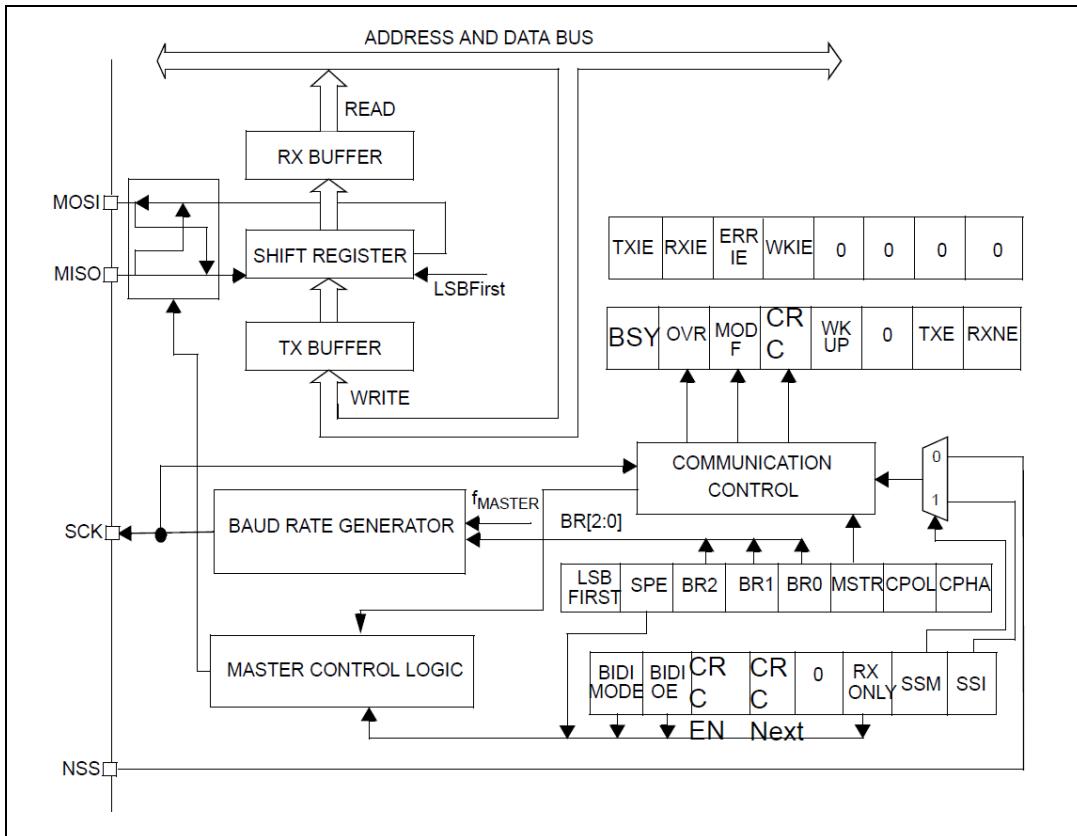
- Full duplex synchronous transfers (on 3 lines)
- Simplex synchronous transfers on 2 lines with or without a bidirectional data line
- Master or slave operation
- 8 Master mode frequencies ( $f_{MASTER}/2$  max.)
- Slave mode frequency ( $f_{MASTER}/2$  max.)
- Faster communication - Maximum SPI speed: 10 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
- Hardware CRC feature for reliable communication:
  - CRC value can be transmitted as last byte in Tx mode
  - CRC error checking for last received byte
- 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 SPI general description

The block diagram of the SPI is shown in [Figure 91](#).

**Figure 91. SPI block diagram**



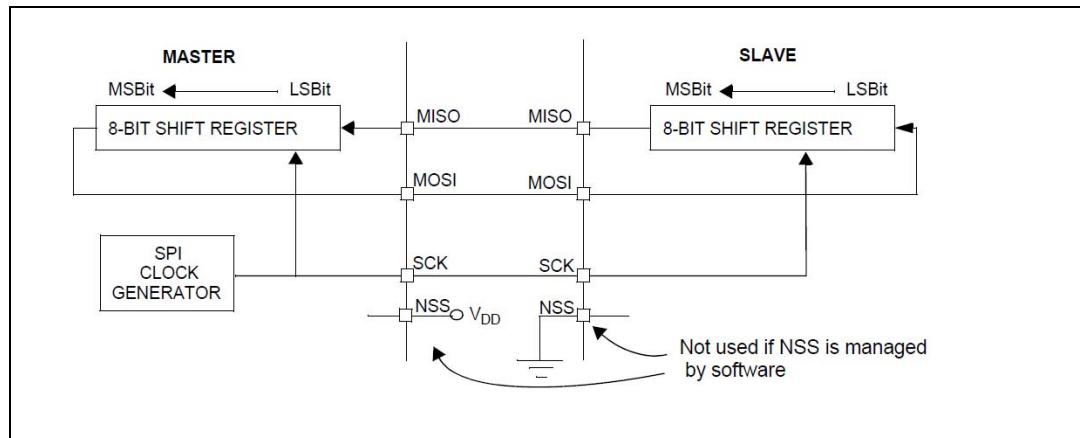
The SPI is connected to external devices through four pins:

- 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 an 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 20.3.9: Error flags on page 279](#)).

A basic example of interconnections between a single master and a single slave is illustrated in [Figure 92](#).

**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 92. 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.

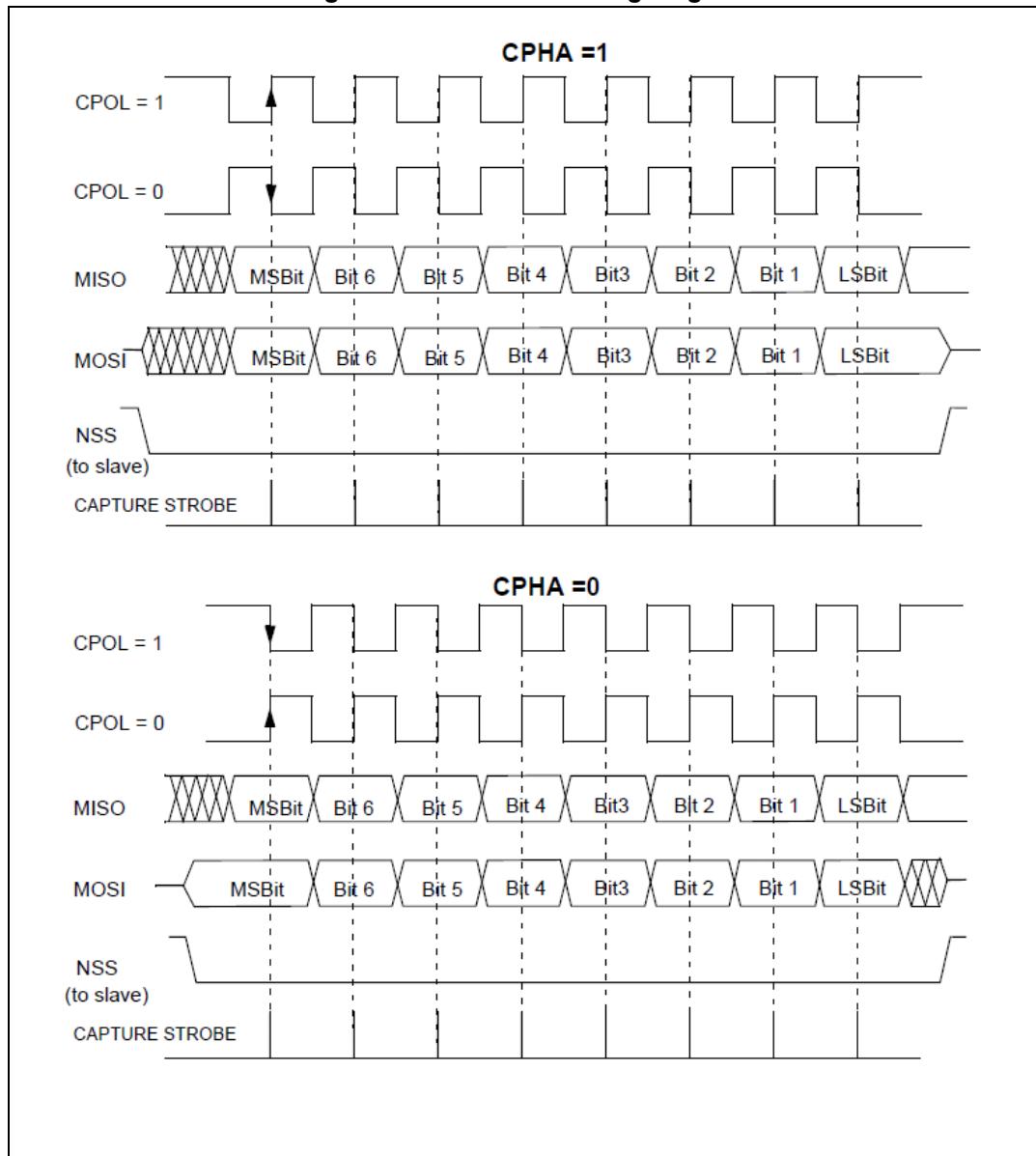
The combination of the CPOL clock polarity and CPHA (clock phase) bits selects the data capture clock edge.

[Figure 93](#) 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 93. 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 93](#)). 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 264](#)), 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 11.8.1: 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 93](#)).
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.

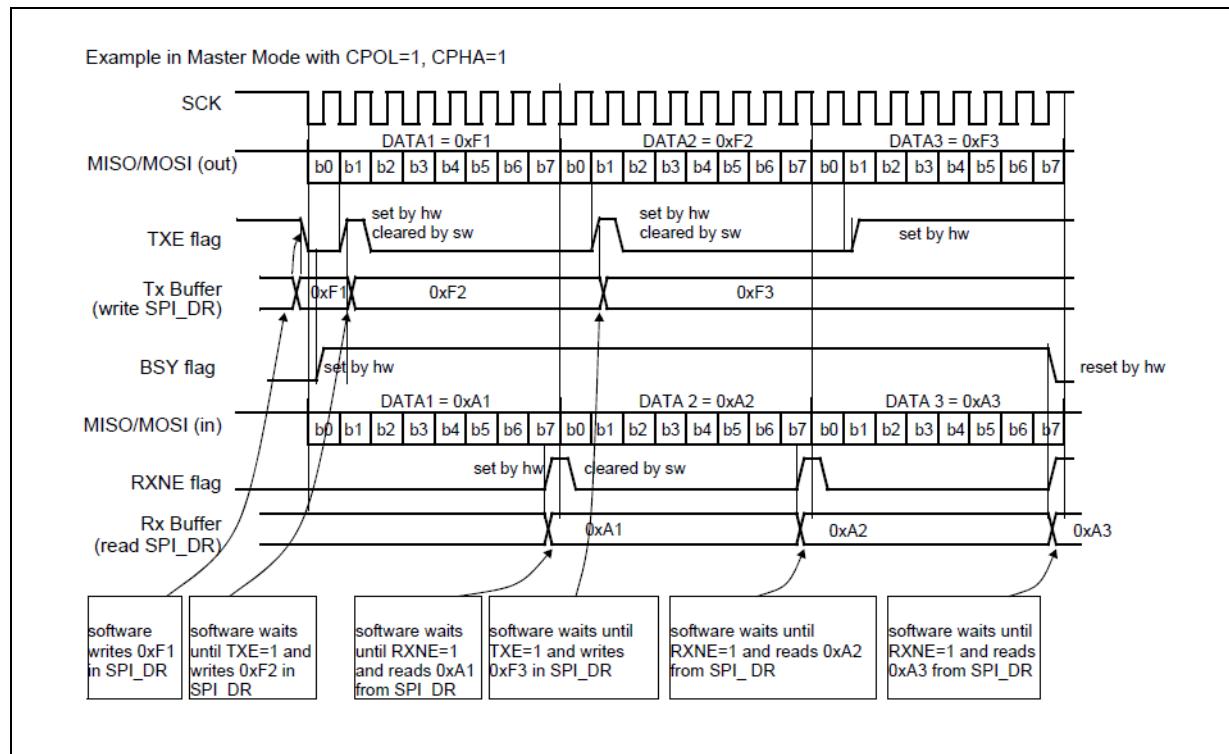
In some configurations, the BSY flag can be used during the last data transfer to wait until the completion of the transfer.

### Full Duplex Transmit and receive procedure in master or slave mode (BDM=0 and RXONLY = 0)

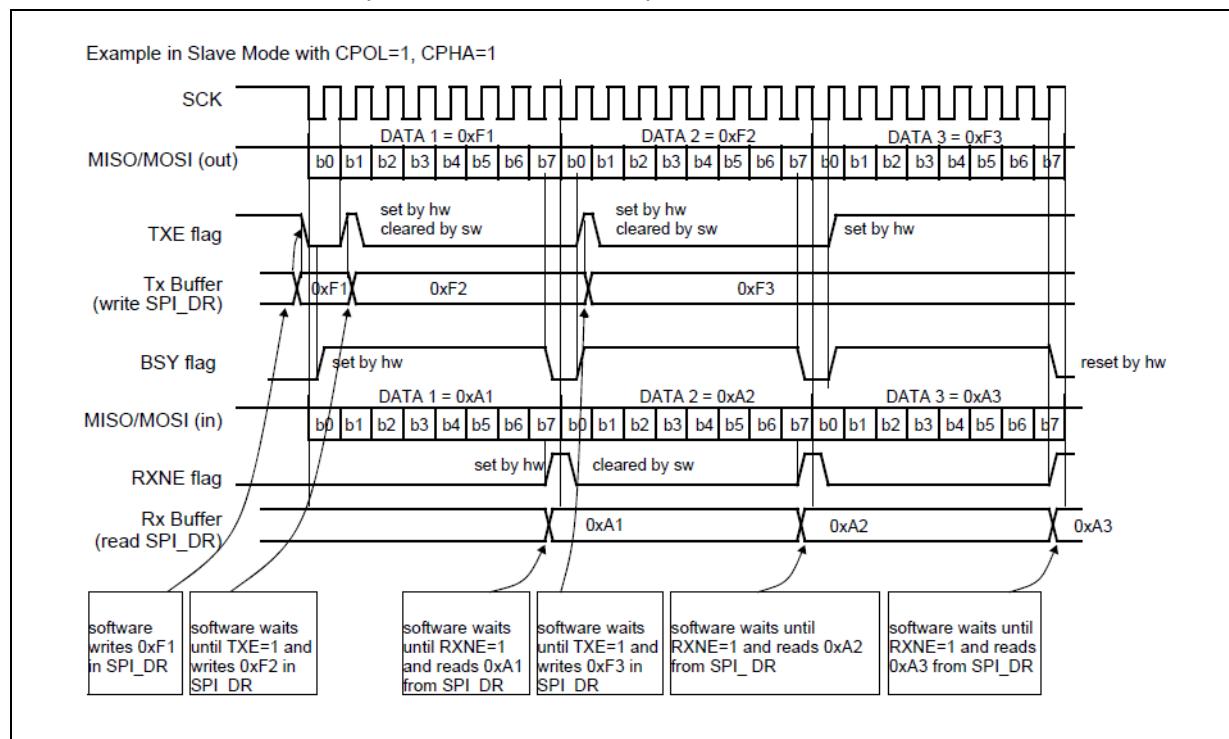
1. Enable the SPI by setting the SPE bit
2. Write the first data to be transmitted in the SPI\_DR register (this clears the TXE flag).
3. Wait until TXE = 1 and write the second data to be transmitted. Then wait until RXNE = 1 and read the SPI\_DR to get the first received data (this clears the RXNE 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 94. TXE/RXNE/BSY behavior in full duplex mode (RXONLY = 0). Case of continuous transfers**



**Figure 95. TXE/RXNE/BSY behavior in slave / full duplex mode (BDM = 0, RXONLY = 0). Case of continuous transfers**



### Transmit-only procedure (BDM = 0 RXONLY = 0)

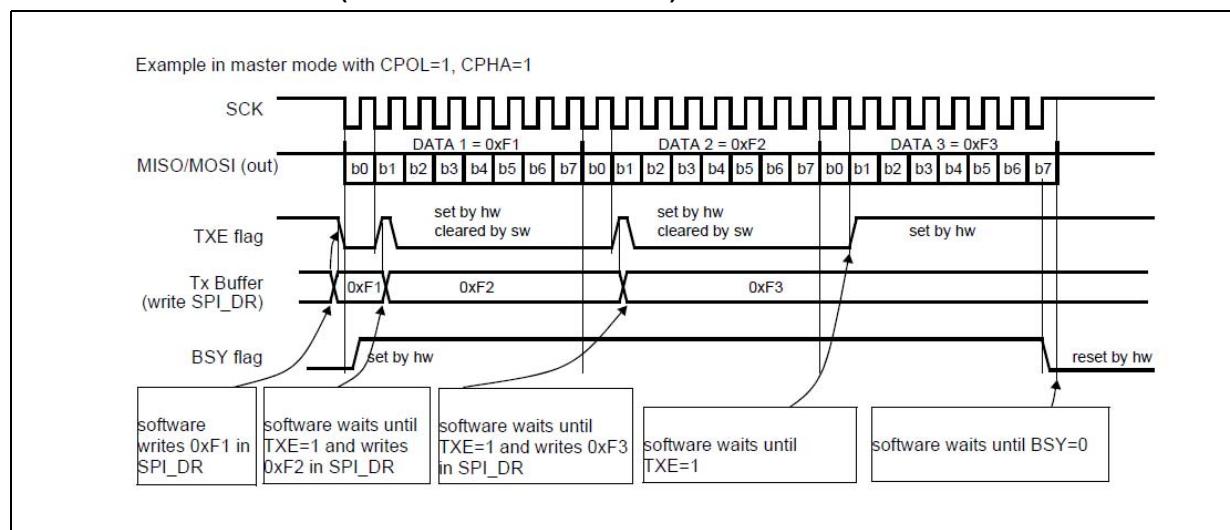
In this mode, the procedure can be reduced as described below and the BSY bit can be used to wait until the effective completion of the transmission (see [Figure 94](#) and [Figure 95](#)):

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.

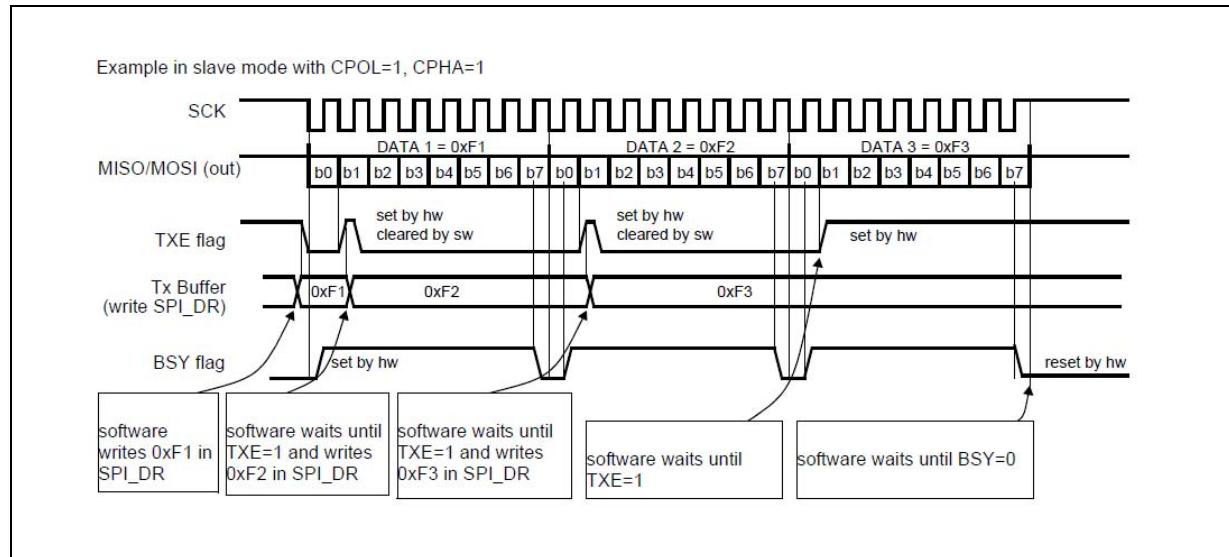
This procedure can be also implemented using dedicated interrupt subroutines launched at each rising edge of TXE flag.

- 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 96. TXE/BSY in master transmit-only mode  
(BDM = 0 and RXONLY = 0). Case of continuous transfers**



**Figure 97. 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)

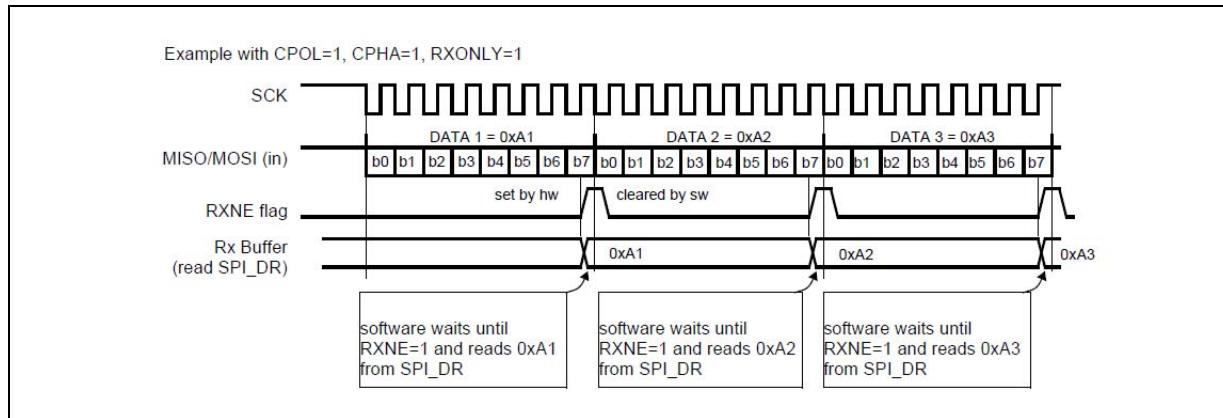
In this mode, the procedure can be reduced as described below (see [Figure 98](#)):

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 20.3.8: Disabling the SPI on page 277](#).

**Figure 98. 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

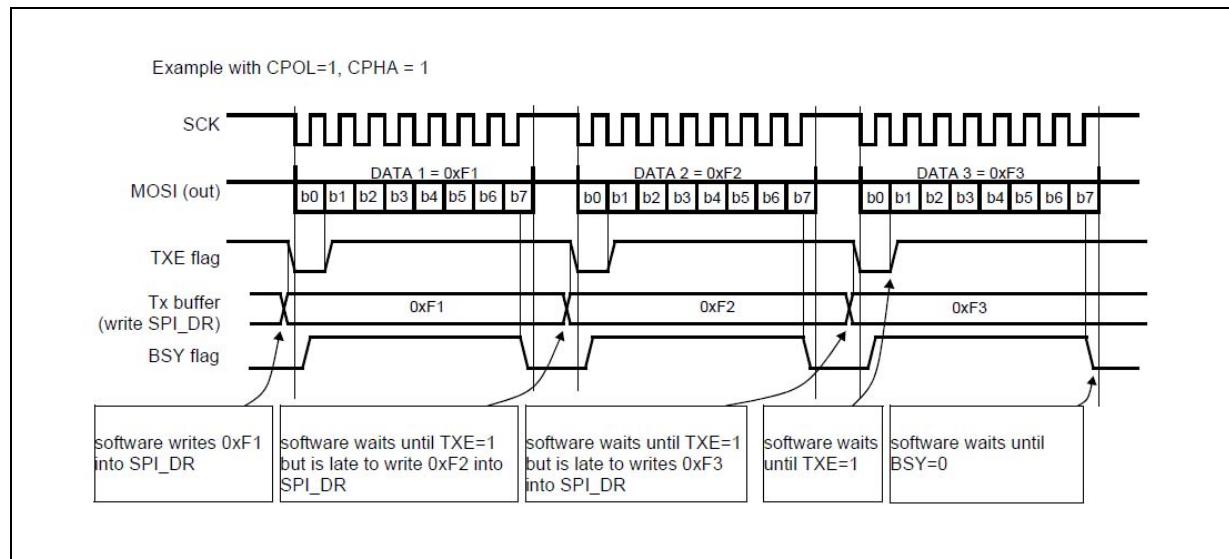
When transmitting data in master mode, if the software is fast enough to detect each TXE rising edge (or TXE interrupt) and to immediately write the SPI\_DR register before the ongoing data transfer is complete, the communication is said to be continuous. In this case, there is no discontinuity in the 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 99](#)).

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 95](#)).

**Figure 99. TXE/BSY behavior when transmitting (BDM = 0 and RXLONY = 0). Case of discontinuous transfers**



### 20.3.6 CRC calculation

A CRC calculator has been implemented for communication reliability. Separate CRC calculators are implemented for transmitted data and received data. The CRC is calculated using a programmable polynomial serially on each bit. The CRC is calculated on the sampling clock edge defined by the CPHA and CPOL bits in the SPI\_CR1 register.

CRC calculation is enabled by setting the CRCEN bit in the SPI\_CR1 register. This action resets the CRC registers (SPI\_RXCRCR and SPI\_TXCRCR). When the CRCNEXT bit in SPI\_CR2 is set, the SPI\_TXCRCR value is transmitted at the end of the current byte transmission.

If a byte is present in the Tx buffer, the CRC value is transmitted only after the transmission of this byte. During the transmission of CRC, the CRC calculator is switched off and the register value remains unchanged.

The CRCERR flag in the SPI\_SR register is set if the value received in the shift register during the SPI\_TXCRCR value transmission does not match the SPI\_RXCRCR value.

SPI communication using CRC is possible through the following procedure:

- Program the CPOL, CPHA, LSBfirst, BR, SSM, SSI and MSTR values.
- Program the polynomial in the SPI\_CRCPR register
- Enable the CRC calculation by setting the CRCEN bit in the SPI\_CR1 register. This also clears the SPI\_RXCRCR and SPI\_TXCRCR registers
- Enable the SPI by setting the SPE bit in SPI\_CR1
- Start the communication and sustain the communication until all but one byte has been transmitted or received.
- On writing the last byte to the Txbuffer, set the CRCNext bit in the SPI\_CR2 register to indicate that after transmission of the last byte, the CRC should be transmitted. The CRC calculation will be frozen during the CRC transmission.
- After transmitting the last byte, the SPI transmits the CRC. CRCNext bit is reset. The CRC is also received and compared against the SPI\_RXCRCR value. If the value does not match, the CRCERR flag in SPI\_SR is set and an interrupt can be generated when the ERRIE in the SPI\_ICR register is set.

**Note:**

*With high bit rate frequencies, the user must take care when transmitting CRC. As the number of used CPU cycles has to be as low as possible in the CRC transfer phase, the calling of software functions in the CRC transmission sequence is forbidden to avoid errors in the last data and CRC reception.*

### 20.3.7 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.*

### 20.3.8 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 ongoing, 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).

### 20.3.9 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.

#### CRC error

This flag is used to verify the correctness of the value received when the CRCEN bit in the SPI\_CR2 register is set. The CRCERR flag in the SPI\_SR register is set if the value received in the shift register after the SPI\_TXCRCR value transmission does not match the SPI\_RXCRCR value. Refer to [Chapter 20.3.6: CRC calculation](#).

### 20.3.10 SPI low power modes

**Table 45. 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.

### 20.3.11 SPI interrupts

**Table 46. 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

**Table 46. SPI interrupt requests (continued)**

Interrupt event	Event flag	Enable control bit	Exit from Wait	Exit from Halt
Wakeup event flag	WKUP	WKIE	Yes	Yes
Master mode fault event	MODF	ERRIE	Yes	No
Overrun error	OVR		Yes	No
CRC error flag	CRCERR		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	C POL	CPHA	
rw	rw		rw	rw	rw	rw	

Bit 7 **LSBFIRST**: Frame format <sup>(1)</sup>

0: MSB is transmitted first

1: LSB is transmitted first

Bit 6 **SPE**: SPI enable <sup>(2)</sup>

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 <sup>(1)</sup>

0: Slave configuration

1: Master configuration

Bit1 **C POL**: Clock polarity <sup>(1)</sup>

0: SCK to 0 when idle

1: SCK to 1 when idle

Bit 0 **CPHA**: Clock phase <sup>(1)</sup>

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 20.3.8: Disabling the SPI on page 277](#)

## 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	CRCEN	CRCNEXT	Reserved	RXOnly	SSM	SSI

**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.

**Bit 5 CRCEN:** Hardware CRC calculation enable

0: CRC calculation disabled

1: CRC calculation Enabled

*Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation*

**Bit 4 CRCNEXT:** Transmit CRC next

0: Next transmit value is from Tx buffer

1: Next transmit value is from Tx CRC register

**Bit 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	r

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 (CRCERR, 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	CRCERR	WKUP	Reserved	TXE	RXNE

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 caution: refer to Section 20.3.7: Status flags on page 277 and Section 20.3.8: Disabling the SPI on page 277*

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 **CRCERR**: CRC error flag

0: CRC value received matches the SPI\_RXCRCR value

1: CRC value received does not match the SPI\_RXCRCR value

This flag is set by hardware and cleared by software writing 0.

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.

### 20.4.6 SPI CRC polynomial register (SPI\_CRCPR)

Address offset: 0x05

Reset value: 0x07

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

Bits 7:0 **CRCPOLY[7:0]**: CRC polynomial register

This register contains the polynomial for the CRC calculation.

The CRC polynomial (0x07) is the reset value of this register. You can configure an other polynomial as required for your application.

### 20.4.7 SPI Rx CRC register (SPI\_RXCRCR)

Address offset: 0x06Reset value: 0x00

7	6	5	4	3	2	1	0
RxCRC[7:0]							
r							

Bits 7:0 **RxCRC[7:0]**: Rx CRC Register

When CRC calculation is enabled, the RxCRC[7:0] bits contain the computed CRC value of the subsequently received bytes. This register is reset when the CRCEN bit in SPI\_CR2 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI\_CRCPR register.

*Note: A read to this register when the BSY Flag is set could return an incorrect value.*

### 20.4.8 SPI Tx CRC register (SPI\_TXCRCR)

Address offset: 0x07 Reset value: 0x00

7	6	5	4	3	2	1	0
TxCRC[7:0]							
r							

Bits 7:0 **TxCRC[7:0]**: Tx CRC register

When CRC calculation is enabled, the TxCRC[7:0] bits contain the computed CRC value of the subsequently transmitted bytes. This register is reset when the CRCEN bit of SPI\_CR2 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI\_CRCPR register.

*Note: A read to this register when the BSY flag is set could return a incorrect value*

## 20.5 SPI register map and reset values

Table 47. 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	CRCEN 0	CRCNEXT 0	- 0	RXONLY 0	SSM 0	SSI 0
0x03	SPI_SR Reset value	BSY 0	OVR 0	MODF 0	CRCERR 0	WKUP 0	- 0	TXE 1	RXNE 0
0x04	SPI_DR Reset value	DR[7:0] 0							
0x05	SPI_CRCPR reset value	MSB 0	- 0	- 0	- 0	- 0	- 1	- 1	LSB 1
0x06	SPI_RXCRCR reset value	MSB 0	- 0	- 0	- 0	- 0	- 0	- 0	LSB 0
0x07	SPI_TXCRCR reset value	MSB 0	- 0	- 0	- 0	- 0	- 0	- 0	LSB 0

## 21 Inter-integrated circuit (I2C) interface

### 21.1 I2C introduction

I2C (inter-integrated circuit) bus interface serves as an interface between the microcontroller and the serial I2C bus. It provides multi-master capability, and controls all I2C bus-specific sequencing, protocol, arbitration and timing. It supports standard and fast speed modes.

### 21.2 I2C main features

- Parallel-bus/I2C protocol converter
- Multi-master capability: the same interface can act as Master or Slave
- I2C Master features:
  - Clock generation
  - Start and Stop generation
- I2C Slave features:
  - Programmable I2C Address detection
  - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and general call
- Supports different communication speeds:
  - Standard speed (up to 100 kHz),
  - Fast speed (up to 400 kHz)
- Status flags:
  - Transmitter/receiver mode flag
  - End-of-byte transmission flag
  - I2C 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

## 21.3 I2C 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 I2C 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) I2C 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

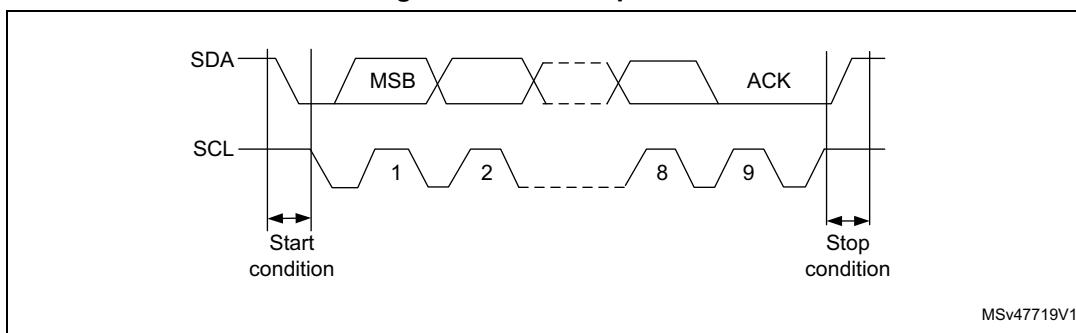
In Master mode, the I2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7- or 10-bit), and the General Call address. The General Call address detection 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.

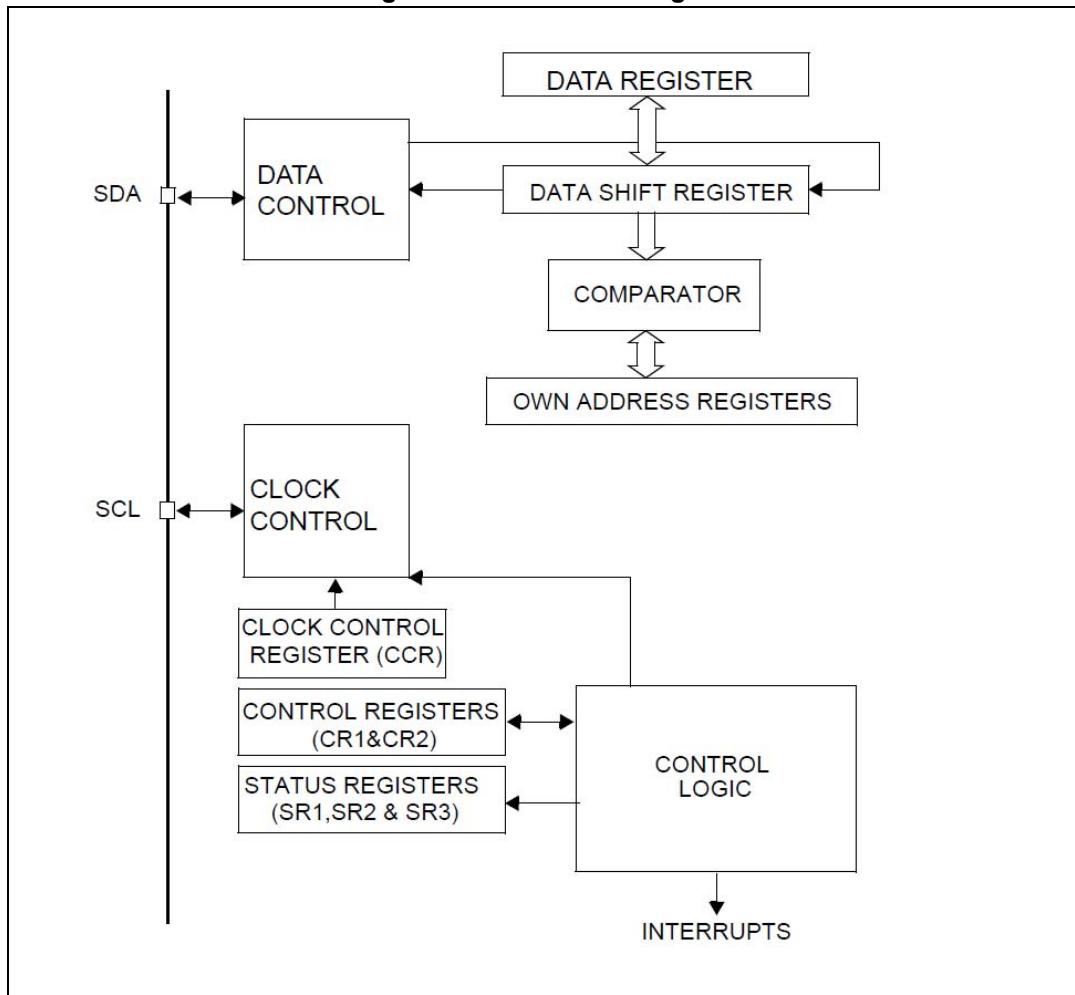
**Figure 100. I2Cbus protocol**



Acknowledge may be enabled or disabled by software. The I2C interface addresses (7-/10-bit and/or general call address) can be selected by software.

The block diagram of the I2C interface is shown in [Figure 101](#).

Figure 101. I2C block diagram



## 21.4 I2C functional description

By default the I2C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

### 21.4.1 I2C 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.

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.

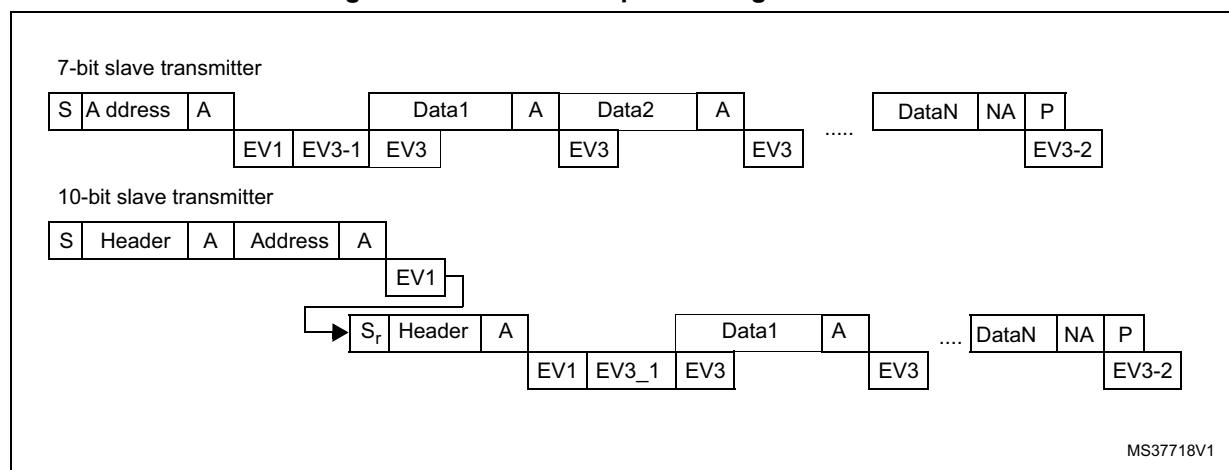
The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see Transfer sequencing EV1 EV3 in the following figure).

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 102. Transfer sequence diagram for slave transmitter**



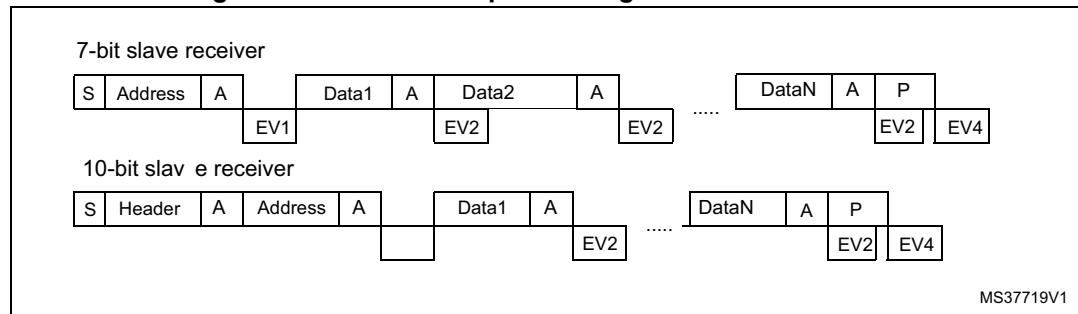
1. Legend:  
**S**= Start, **S<sub>r</sub>**= Repeated Start, **P**= Stop, **A**= Acknowledge, **NA**= Non-acknowledge, **EVx**= Event (with interrupt if ITEVTEN=1)  
**EV1**: ADDR =1, cleared by reading SR1 register followed by reading SR3.  
**EV3-1**: TXE=1, shift register empty, data register empty, 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 software sequence must be performed before the end of the current byte transfer. In case EV3 software sequence can not be managed before the end of the current byte transfer, it is recommended to use BTF instead of TXE with the drawback of slowing the communication.

### 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 103](#)).

**Figure 103. Transfer sequence diagram for slave receiver**

1. Legend:  
**S**= Start, **S<sub>r</sub>** = 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 software sequence must be performed before the end of the current byte transfer.
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 311](#).

### 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 (see [Figure 104](#)).

#### 21.4.2 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<sub>IH</sub> 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 () .

### 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 104](#) & [Figure 105](#) Transfer sequencing EV9).

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 104](#) & [Figure 105](#) Transfer sequencing EV6).

- 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 104](#) & [Figure 105](#) Transfer sequencing EV6).

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 104 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.

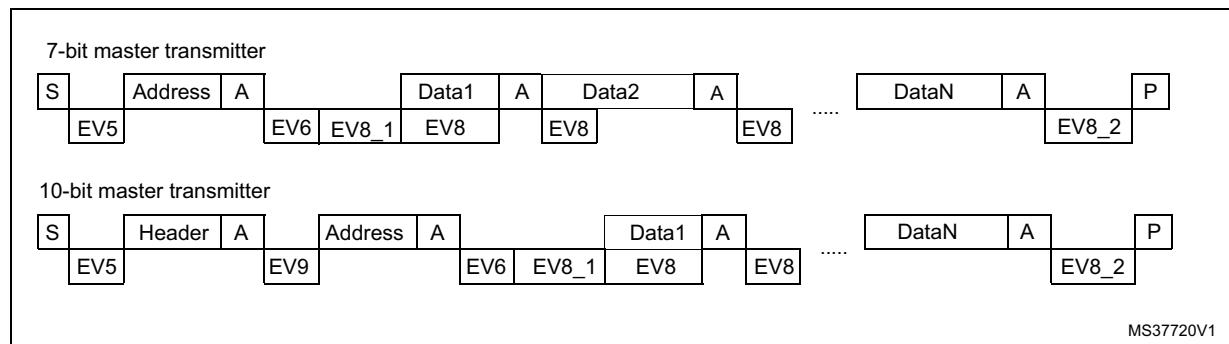
If TXE is set and a data byte was not written in the DR register before the end of the next data transmission, BTF is set and the interface waits until BTF is cleared, by reading the SR1 register and then writing to the DR register, stretching SCL low.

## 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 104 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.*

**Figure 104. Transfer sequence diagram for master transmitter**



1. Legend:  
**S**= Start, **S<sub>r</sub>**= 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 311](#)
2. EV8 software sequence must be performed before the end of the current byte transfer. In case EV8 software sequence can not be managed before the end of the current byte transfer, it is recommended to use BTF instead of TXE with the drawback of slowing the communication.

## Master receiver

Following the address transmission and after clearing ADDR, the I<sup>2</sup>C 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 () .

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.

## Closing the communication

**Method 1:** This method is for the case when the I2C is used with interrupts that have the highest priority in the application.

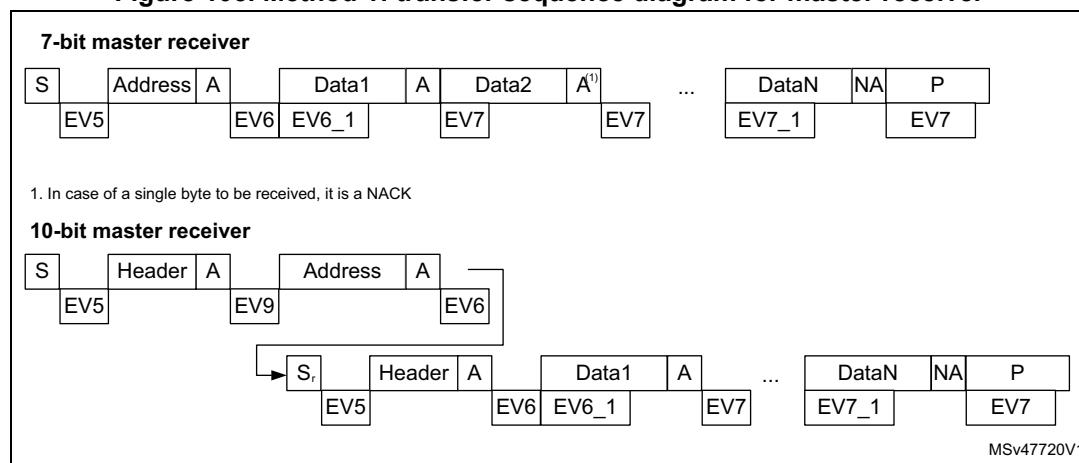
The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a Stop/Re-Start condition.

- 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).

**Method 1:** This method is for the case when the I2C is used with interrupts that have the highest priority in the application.

Figure 105. Method 1: transfer sequence diagram for master receiver



1. Legend:  
**S**= Start, **S<sub>r</sub>** = 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 the DR and shift registers are full, the next data reception ( $I^2C$  clock generation for slave) is performed after the EV7 event is cleared. In this case, EV7 does not overlap with data reception.
3. If a single byte is received, it is NA.
4. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
5. EV7 software sequence must be completed before the end of the current byte transfer. In case EV7 software sequence can not be managed before the current byte end of transfer, it is recommended to use BTF instead of RXNE with the drawback of slowing the communication.
6. The EV6\_1 or EV7\_1 software sequence must be completed before the ACK pulse of the current byte transfer.
7. See also: [Note 8 on page 311](#).

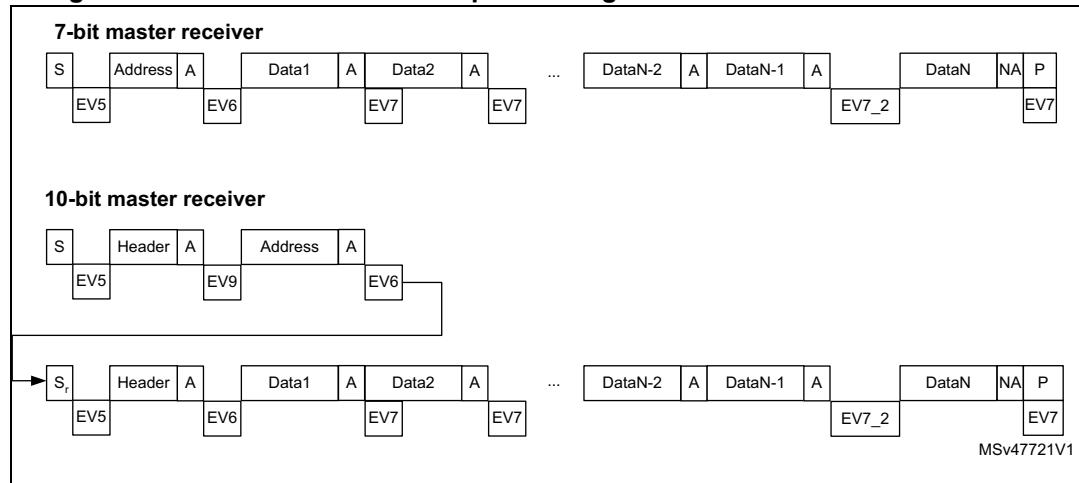
**Method 2:** This method is for the case when the I2C is used with interrupts that do not have the highest priority in the application or when the I2C is used with polling.

With this method:

- DataN\_2 is not read, so that after DataN\_1, the communication is stretched (both RxNE and BTF are set).
- Then, the ACK bit must be cleared before reading DataN-2 in DR to make sure this bit has been cleared before the DataN Acknowledge pulse.
- After that, just after reading DataN\_2, software must set the STOP/ START bit and read DataN\_1. After RxNE is set, read DataN.

This is illustrated in the following figure:

**Figure 106. Method 2: transfer sequence diagram for master receiver when N > 2**



1. Legend:

- S**= Start, **S<sub>r</sub>** = 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 the DR register.  
**EV6:** ADDR1, 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.  
**EV7:** RxNE=1, cleared by reading DR register.  
**EV7\_2:** BTF = 1, DataN-2 in DR and DataN-1 in shift register, program ACK = 0, Read DataN-2 in DR.  
Program STOP = 1, read DataN-1.

**EV9:** ADD10= 1, cleared by reading SR1 register followed by writing DR register.

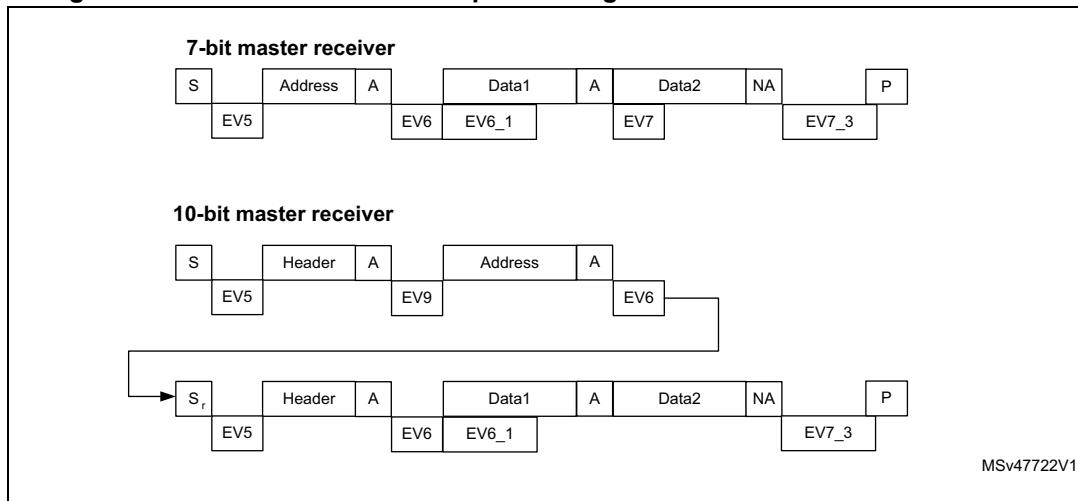
2. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
3. EV7 software sequence must be completed before the end of the current byte transfer. In case EV7 software sequence can not be managed before the current byte end of transfer, it is recommended to use BTF instead of RXNE, with the drawback of slowing the communication.

When 3 bytes remain to be read:

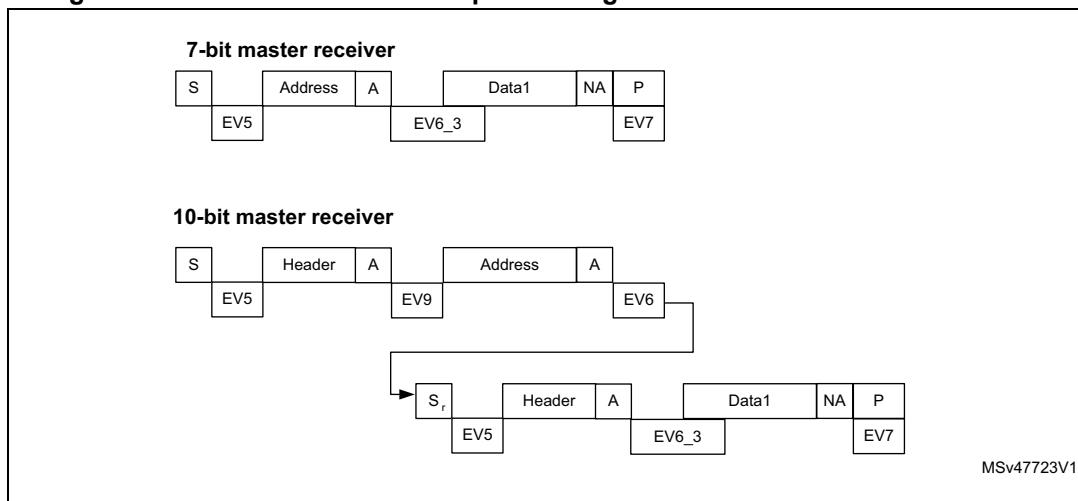
- RxNE = 1 => Nothing (DataN-2 not read).
- DataN-1 received
- BTF = 1 because both shift and data registers are full: DataN-2 in DR and DataN-1 in the shift register => SCL tied low: no other data will be received on the bus.
- Clear ACK bit
- Read DataN-2 in DR => This launches the DataN reception in the shift register
- DataN received (with a NACK)
- Program START/STOP
- Read DataN-1
- RxNE = 1
- Read DataN

The procedure described above is valid for N>2. The cases where a single byte or two bytes are to be received should be handled differently, as described below:

- Case of a single byte to be received:
  - In the ADDR event, clear the ACK bit.
  - Clear ADDR
  - Program the STOP/START bit.
  - Read the data after the RxNE flag is set.
- Case of two bytes to be received:
  - Set POS and ACK
  - Wait for the ADDR flag to be set
  - Clear ADDR
  - Clear ACK
  - Wait for BTF to be set
  - Program STOP
  - Read DR twice

**Figure 107. Method 2: transfer sequence diagram for master receiver when N=2**

1. Legend:  
**S**= Start, **S<sub>r</sub>** = 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 the DR register.  
**EV6**: ADDR1, 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. The acknowledge should be disabled just after EV6, that is after ADDR is cleared  
**EV7\_3**: BTF = 1, program STOP = 1, read DR twice (Read Data1 and Data2) just after programming the STOP.  
**EV9**: ADD10= 1, cleared by reading SR1 register followed by writing DR register.
2. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
3. EV6\_1 software sequence must be completed before the ACK pulse of the current byte transfer.

**Figure 108. Method 2: transfer sequence diagram for master receiver when N=1**

1. Legend:  
**S**= Start, **S<sub>r</sub>** = 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 the DR register.  
**EV6**: ADDR =1, cleared by reading SR1 register followed by reading SR3 register.  
**EV6\_3**: ADDR = 1, program ACK = 0, clear ADDR by reading SR1 register followed by reading SR3 register, program STOP =1 just after ADDR is cleared.  
**EV7**: RxNE =1, cleared by reading DR register.

- EV9:** ADD10= 1, cleared by reading SR1 register followed by writing DR register.
2. EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
  3. EV6\_3 software sequence must be completed before the ACK pulse of the current byte transfer.

### 21.4.3 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 I<sup>2</sup>C 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<sup>2</sup>C 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.

#### 21.4.4 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.

## 21.5 I2C low power modes

Table 48. I2C interface behavior in low power modes<sup>(1)</sup>

Mode	Description
Wait	No effect on I2C interface. I2C interrupts cause the device to exit from Wait mode.
Halt	<p><b>In slave mode:</b> Communication is reset, except for configuration registers. Device is in slave mode.</p> <p>Wakeup from Halt interrupt is generated if ITEVTEN = 1 and address matched (including allowed headers).</p> <p>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.</p> <p>If NOSTRETCH = 0, SCLH will be stretched after acknowledge pulse in Halt mode until WUFH is cleared by software;</p> <p>None of the flags are set by the address which wakes up the CPU.</p> <p><b>In master mode:</b> 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.</p> <p><b>Note:</b> <i>It is forbidden to enter Halt mode while a communication is ongoing.</i></p>

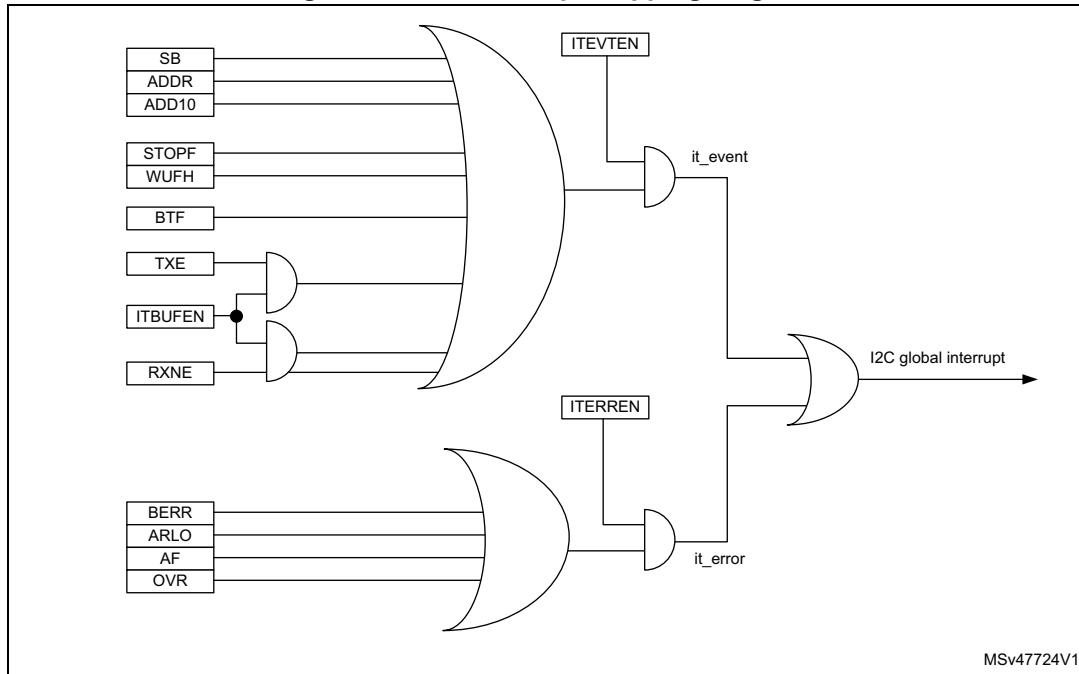
1. I2C cannot operate at low power run and low power wait modes due to low PCLK frequency.

## 21.6 I2C interrupts

Table 49. I2C Interrupt requests

Interrupt event	Event flag	Enable control bit	Exit from Wait	Exit from Halt
Start bit sent (Master)	SB			
Address sent (Master) or Address matched (Slave)	ADDR	ITEVTEN		No
10-bit header sent (Master)	ADD10			
Stop received (Slave)	STOPF			
Data byte transfer finished	BTF			
Wakeup from Halt	WUFH	ITEVTEN	Yes	Yes
Receive buffer not empty	RXNE	ITEVTEN and ITBUFEN		
Transmit buffer empty	TXE			
Bus error	BERR			
Arbitration loss (Master)	ARLO			
Acknowledge failure	AF			
Overrun/underrun	OVR	ITERREN		No

Figure 109. I2C interrupt mapping diagram



## 21.7 I2C registers

### 21.7.1 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			r		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.*

## 21.7.2 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		r		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 is used when the procedure for reception of 2 bytes (see Method 2: transfer sequence diagram for master receiver when N=2) is followed. It must be configured before data reception starts. In this case, to NACK the 2nd byte, the ACK bit must be cleared just after ADDR is cleared.

*Note:*

### 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 or START request may occur.

### 21.7.3 Frequency register (I2C\_FREQR)

Address offset: 0x02

Reset value: 0x00

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

Bits 7:6 Reserved

Bits 5:0 **FREQ[5:0]** Peripheral clock frequency. <sup>(1)</sup>

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 24 MHz

000000: not allowed

000001: 1 MHz

000010: 2 MHz

...

011000: 24 MHz

Higher values: not allowed

1. The minimum peripheral clock frequencies for respecting the I<sup>2</sup>C bus timings are:  
1 MHz for standard mode and 4 MHz for fast mode

### 21.7.4 Own address register LSB (I2C\_OARL)

Address offset: 0x03

Reset value: 0x00

7	6	5	4	3	2	1	0
ADD[7:1]							ADD0
rw							rw

Bits 7:1 **ADD[7:1]** Interface address

bits 7:1 of address

Bit 0 **ADD[0]** Interface address

7-bit addressing mode: don't care

10-bit addressing mode: bit 0 of address

### 21.7.5 Own address register MSB (I2C\_OARH)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
ADDMODE	ADDCONF		Reserved		ADD[9:8]		Reserved
rw	rw		r		rw		r

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 **ADD[9:8]** Interface address

10-bit addressing mode: bits 9:8 of address.

Bit 0 Reserved

### 21.7.6 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 <sup>(1)(2)(3)</sup>

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.

### 21.7.7 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	r

Bit 7 **TXE**: Data register empty (transmitters)<sup>(1)</sup>

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)<sup>(2)(3)</sup>

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:* RXNE 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)<sup>(4)(5)</sup>

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)<sup>(6)</sup>

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<sup>(7)(8)</sup>

- 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)<sup>(8)(9)</sup>

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)<sup>(8)</sup>

- 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 103: Transfer sequence diagram for slave receiver on page 293](#)
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 fMASTER = fCPU 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 fMASTER = fCPU = 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 103: Transfer sequence diagram for slave receiver on page 293](#).

### 21.7.8 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	
r	rc_w0	r	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.

### 21.7.9 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	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.*

### 21.7.10 Interrupt register (I2C\_ITR)

Address offset: 0x0A

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved				ITBUFEN	ITEVTEN	ITERREN	
r				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

### 21.7.11 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:

$$\text{Period(I2C)} = 2 * \text{CCR} * t_{\text{MASTER}}$$

$$t_{\text{high}} = \text{CCR} * t_{\text{MASTER}}$$

$$t_{\text{low}} = \text{CCR} * t_{\text{MASTER}}$$

- Fast mode:

If DUTY = 0:

$$\text{Period(I2C)} = 3 * \text{CCR} * t_{\text{MASTER}}$$

$$t_{\text{high}} = \text{CCR} * t_{\text{MASTER}}$$

$$t_{\text{low}} = 2 * \text{CCR} * t_{\text{MASTER}}$$

If DUTY = 1: (to reach 400 kHz)

$$\text{Period(I2C)} = 25 * \text{CCR} * t_{\text{MASTER}}$$

$$t_{\text{high}} = 9 * \text{CCR} * t_{\text{MASTER}}$$

$$t_{\text{low}} = 16 * \text{CCR} * t_{\text{MASTER}}$$

Note:  $t_{CK} = 1/f_{\text{MASTER}}$ .  $f_{\text{MASTER}}$  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.

$t_{\text{high}} = t_{r(\text{SCL})} + t_{w(\text{SCLH})}$ . See device datasheet for the definitions of parameters.

$t_{\text{low}} = t_{f(\text{SCL})} + t_{w(\text{SCLL})}$ . See device datasheet for the definitions of parameters.

I2C communication speed,  $f_{\text{SCL}} = 1/(t_{\text{high}} + t_{\text{low}})$

The real frequency may differ due to the analog noise filter input delay.

### 21.7.12 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	r			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)

Controls the SCLH clock in master mode.

- Standard mode:

$$\text{Period(I2C)} = 2 * \text{CCR} * t_{MASTER}$$

$$t_{high} = \text{CCR} * t_{MASTER}$$

$$t_{low} = \text{CCR} * t_{MASTER}$$

- Fast mode:

If DUTY = 0:

$$\text{Period(I2C)} = 3 * \text{CCR} * t_{MASTER}$$

$$t_{high} = \text{CCR} * t_{MASTER}$$

$$t_{low} = 2 * \text{CCR} * t_{MASTER}$$

If DUTY = 1: (to reach 400 kHz)

$$\text{Period(I2C)} = 25 * \text{CCR} * t_{MASTER}$$

$$t_{high} = 9 * \text{CCR} * t_{MASTER}$$

$$t_{low} = 16 * \text{CCR} * t_{MASTER}$$

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 =>  $40 \times 125$  ns = 5000 ns.)

*Note:  $t_{high} = t_{f(SCL)} + t_{w(SCLH)}$ . 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.*

**Note:** The CCR registers must be configured only when the I<sup>2</sup>C 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 50. I2C\_CCR values for SCL frequency table (fMASTER = 10 MHz or 16 MHz<sup>(1)</sup>)**

I2C Speed	I2C frequency (f <sub>SCL</sub> )	fMASTER = 10 MHz				fMASTER= 16 MHz			
	in Hz	Actual (Hz)	% Error (%)	I2C_CCR (h)	Duty cycle bit	Actual (Hz)	% Error (%)	I2C_CCR (h)	Duty cycle bit
Fast speed	400000	400000	0	1	1	410256.41	2.56	D	0
	370000	370370.37	0.10	9	0	380952.38	2.96	E	0
	350000	370370.37	5.82	9	0	355555.56	1.59	F	0
	320000	333333.33	4.17	A	0	320000	0	2	1
	300000	303030.30	1.01	B	0	313725.49	4.57	11	0
	270000	277777.78	2.88	C	0	280701.75	3.96	13	0
	250000	256410.26	2.56	D	0	253968.25	1.59	15	0
	220000	222222.22	1.01	F	0	222222.22	1.01	18	0
	200000	200000	0	2	1	205128.20	2.56	1A	0
	170000	175438.60	3.20	13	0	172043.01	1.20	1F	0
	150000	151515.15	1.01	16	0	152380.95	1.59	23	0
	120000	123456.79	2.88	1B	0	121212.12	1.01	2C	0
Standard speed	100000	100000	0	32	No impact	100000	0	50	No impact
	50000	50000	0	64		50000	0	A0	
	30000	30120.48	0.40	A6		30075.19	0.25	10A	
	20000	20000	0	FA		20000	0	190	

- The following table gives the values to be written in the I2C\_CCR register to obtain the required I<sup>2</sup>C SCL line frequency

### 21.7.13 TRISE register (I2C\_TRISER)

Address offset: 0x0D

Reset value: 0x02

7	6	5	4	3	2	1	0		
Reserved				TRISE[5:0]					
r				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.

( $1000\text{ ns} / 125\text{ ns} = 8 + 1$ )

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).*

### 21.7.14 I2C register map and reset values

Table 51. I2C 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_OARL Reset value	ADD[7:1] 0000000							ADD0 0		
0x04	I2C_OARH Reset value	ADDMODE 0	ADDCONF 0	- 0	- 0	- 0	ADD[9:8] 00		- 0		
0x05	Reserved										
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	- 0	- 0	- 0	GENCALL 0	- 0	TRA 0	BUSY 0	MSL 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							

## 22 Universal asynchronous receiver transmitter (UART)

### 22.1 Introduction

The UARTs in the STM8S and STM8AF microcontroller families (UART1, UART2, UART3 or UART4) offer a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format (UART mode). The STM8 UARTs offer a very wide range of baud rates and can also be used for multi-processor communication. They also support LIN (Local Interconnection Network) protocol version 1.3, 2.0, 2.1, 2.2 and J2602.

UART1, UART2 and UART4 have extended features (see [Table 52](#)):

- LIN slave mode is supported in UART2, UART3 and UART4.
- Synchronous one-way communication, Smartcard Protocol and IrDA (Infrared Data Association) SIR ENDEC specifications are supported in UART1, UART2 and UART4.
- Half-duplex single wire communication is supported in UART1 and UART4.

Refer to the datasheet for information on the availability of the UART configurations (UART1, UART2, UART3 or UART4) in each microcontroller type.

**Table 52. UART configurations<sup>(1)</sup>**

Feature	UART1	UART2	UART3	UART4
Asynchronous mode	X	X	X	X
Multiprocessor communication	X	X	X	X
Synchronous communication	X	X	NA	X
Smartcard mode	X	X	NA	X
IrDA mode	X	X	NA	X
Single-wire Half-duplex mode	X	NA	NA	X
LIN master mode	X	X	X	X
LIN slave mode	NA	X	X	X

1. X = supported; NA = not applicable.

## 22.2 UART 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 fMASTER/16
- Programmable data word length (8 or 9 bits)
- Configurable stop bits
  - Support for 1 or 2 stop bits
- LIN Master mode (UART1, UART2, UART3 and UART4)
  - LIN break and delimiter generation
  - LIN break and delimiter detection with separate flag and interrupt source for readback checking
- LIN Slave mode (UART2, UART3 and UART4)
  - Autonomous header handling and mute mode to filter responses
  - Identifier parity error checking
  - LIN automatic resynchronization, allowing operation with internal RC oscillator (HSI) clock source.
  - Break detection at any time, even during a byte reception
  - Header errors detection:
    - Delimiter too short
    - Synch field error
    - Deviation error (if automatic resynchronization is enabled)
    - Framing error in synch field or identifier field
    - Header timeout
- Transmitter clock output for synchronous communication (UART1, UART2 and UART4)
- IrDA SIR Encoder Decoder (UART1, UART2 and UART4)
  - Support for 3/16 bit duration for normal mode
- Smartcard Emulation Capability (UART1, UART2 and UART4)
  - Smartcard interface supporting the asynchronous protocol for Smartcards as defined in ISO 7816-3 standards
  - 1.5 Stop bits for Smartcard operation
- Single wire Half Duplex Communication (UART1 and UART4)
- 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
- 6 interrupt sources with flags
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle line received
  - Overrun error
  - Framing error or noise flag
- 2 interrupt vectors
  - Transmitter interrupt
  - Receiver interrupt when register is full
- 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

## 22.3 UART functional description

The interface is externally connected to another device by two or three pins (see [Figure 110: UART1 block diagram](#), [Figure 111: UART2 block diagram](#) and [Figure 112: UART3 block diagram](#)). Any UART bidirectional communication requires a minimum of two pins: UART Receive data input (UART\_RX) and UART transmit data output (UART\_TX):

UART\_RX is the serial data input. Over-sampling techniques are used for data recovery by discriminating between valid incoming data and noise.

UART\_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 UART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 1, 1.5 and 2 Stop bits indicating that the frame is complete
- A status register (UART\_SR)
- Data Register (UART\_DR)
- 16-bit baud rate prescaler (UART\_BRR)
- Guard time Register for use in Smartcard mode

Refer to the register description for the definitions of each bit.

The following pin is required to interface in synchronous mode:

**UART\_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.

The UART\_RX and UART\_TX pins are used in IrDA mode as follows:

**UART\_RX** = IrDA\_RDI: Receive Data Input in IrDA mode

**UART\_TX** = IrDA\_TDO: Transmit Data Output in IrDA mode

Figure 110. UART1 block diagram

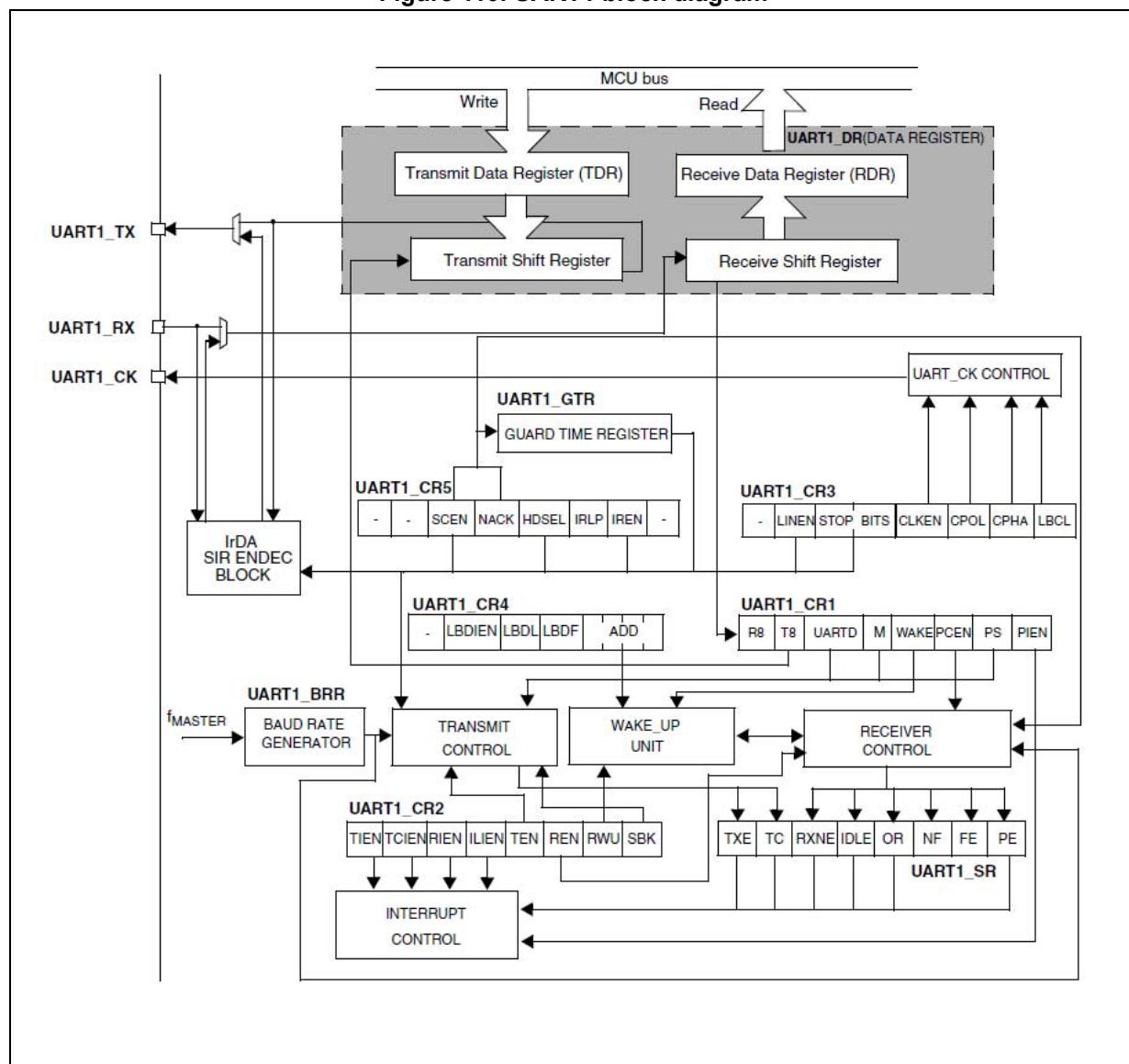


Figure 111. UART2 block diagram

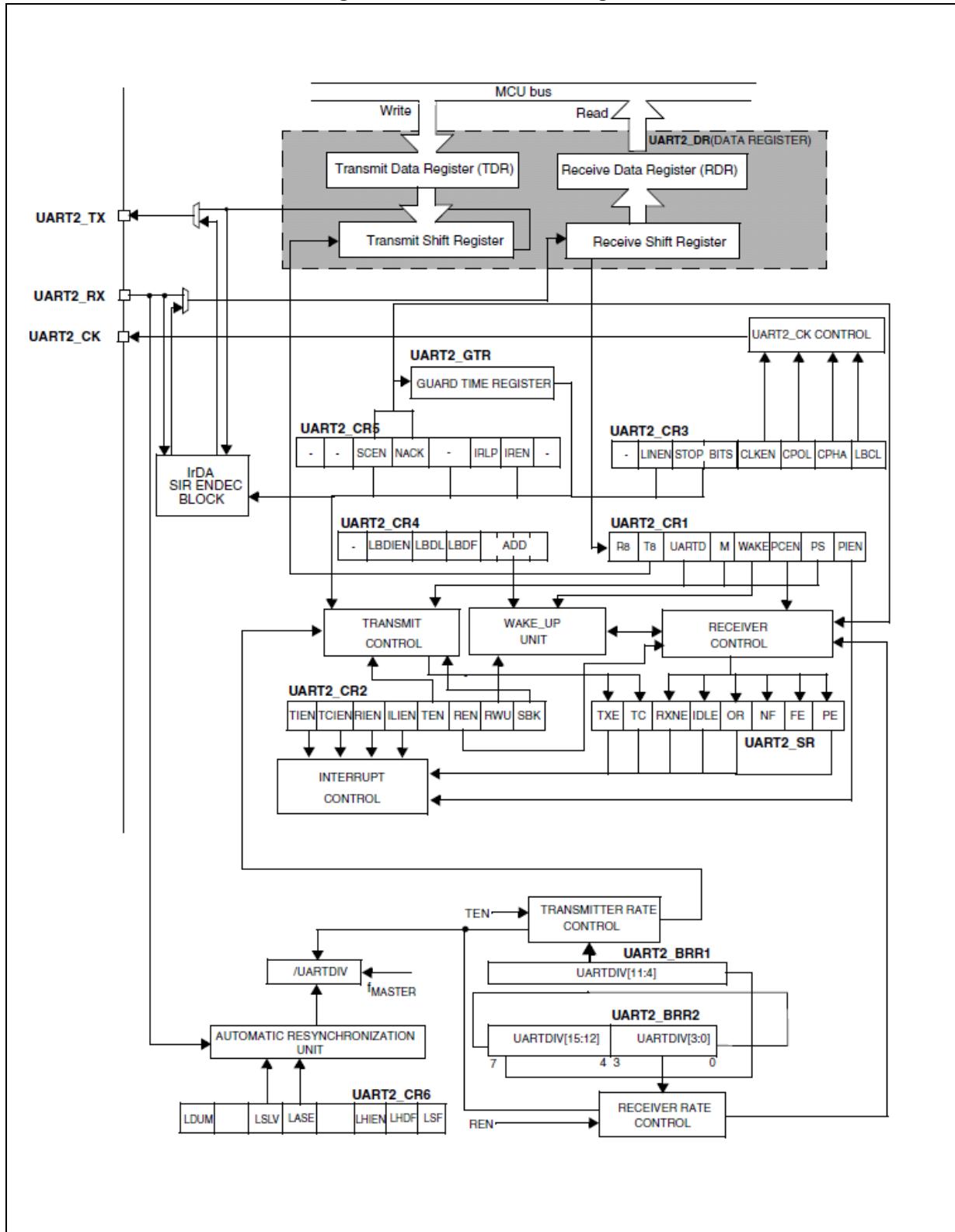


Figure 112. UART3 block diagram

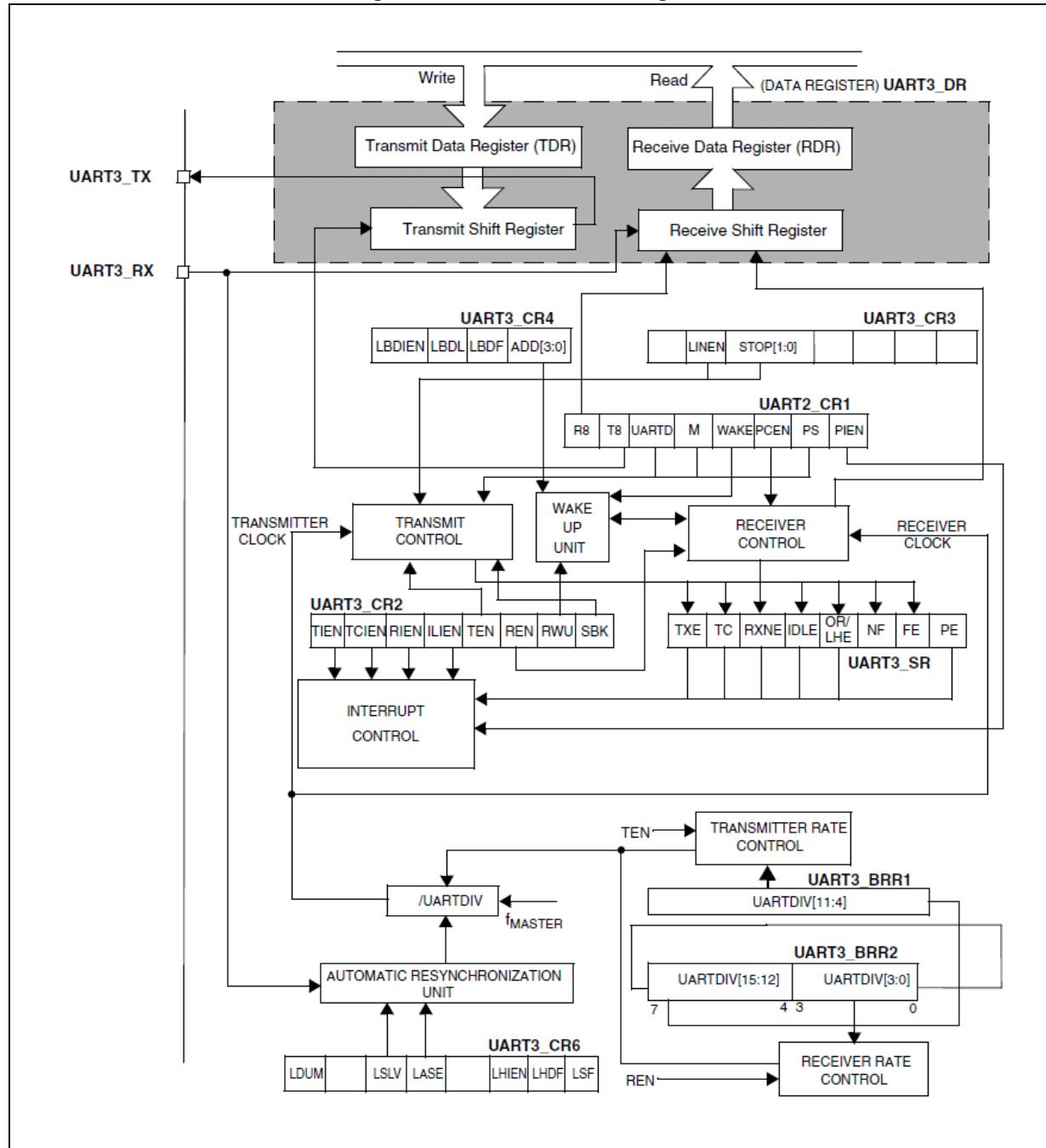
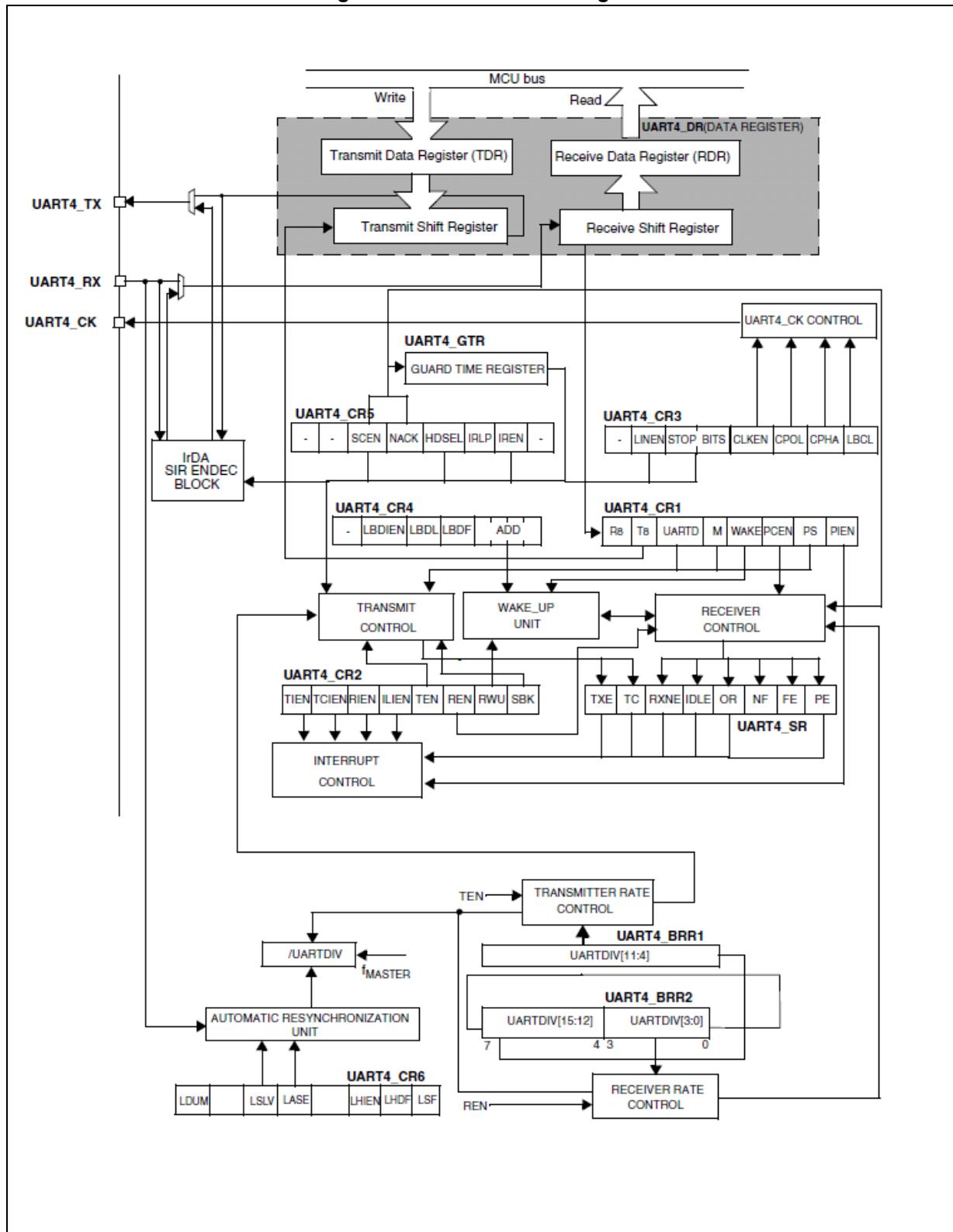


Figure 113. UART4 block diagram



### 22.3.1 UART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the UART\_CR1 register (see [Figure 114](#)).

The UART\_TX pin is in low state during the start bit. It is in high state during the stop bit.

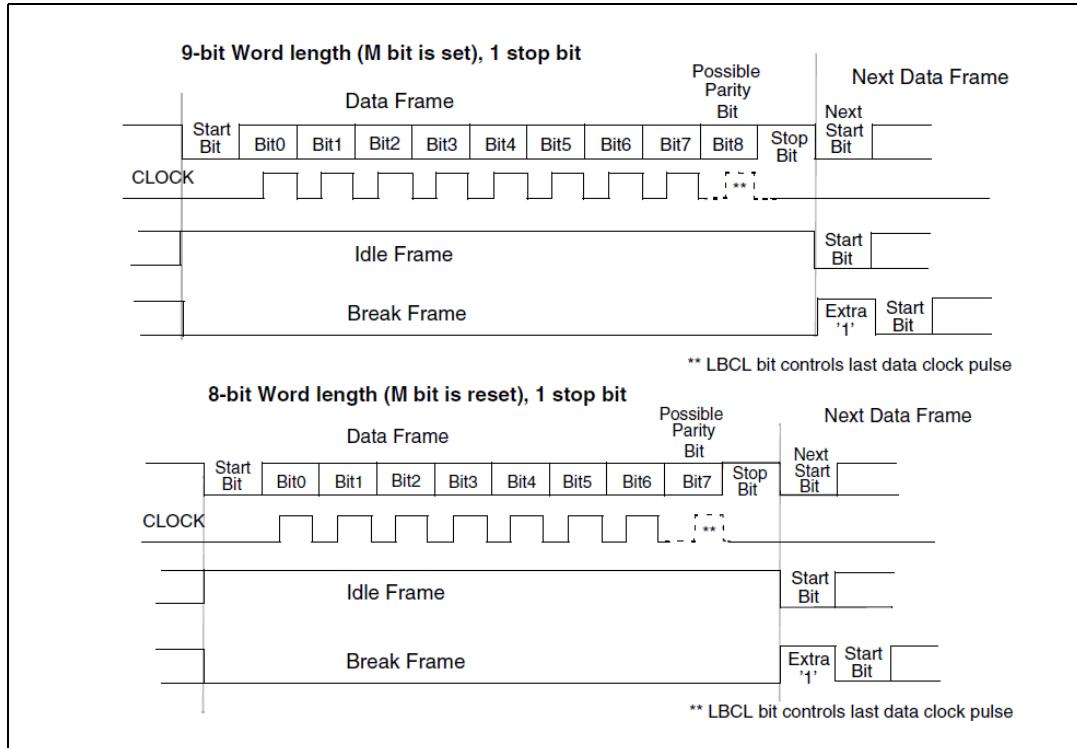
An ***Idle character*** is interpreted as an entire frame of “1”s (the number of “1” ‘s includes the start bit, the number of data bits and the number of stop bits).

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 114. Word length programming**



### 22.3.2 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 UART\_CR1 register.

When the transmit enable bit (TEN) is set, the data in the transmit shift register is output on the UART\_TX pin and the corresponding clock pulses are output on the UART\_CK pin.

#### Character transmission

During an UART transmission, data shifts out least significant bit first on the UART\_TX pin. In this mode, the UART\_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see [Figure 110](#)).

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 UART.

- 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 UART\_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

The number of stop bits to be transmitted with every character can be programmed in Control register 3, bits 5,4.

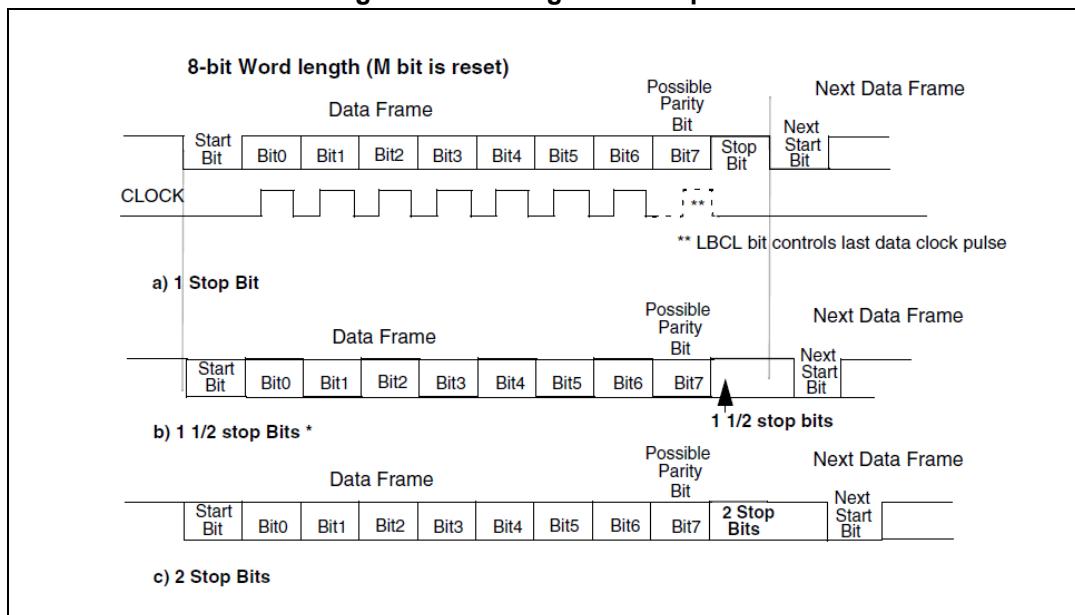
- 1 stop bit: This is the default value of number of stop bits.
- 2 Stop bits: This is supported by normal mode UART.
- 1.5 Stop bits: To be used in Smartcard mode only.

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).

- Note: *In LIN mode (see [Section 22.3.8 on page 340](#)), a standard 13-bit break is always generated.*

Figure 115. Configurable stop bits



## Procedure:

1. Program the M bit in UART\_CR1 to define the word length.
2. Program the number of stop bits in UART\_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
  - a) UART\_BRR2
  - b) UART\_BRR1
4. Set the TEN bit in UART\_CR2 to enable transmitter mode.
5. Write the data to send in the UART\_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 UART\_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 UART 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 UART\_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 UART\_DR register stores the data in the TDR register. The data is copied in the shift register at the end of the current transmission.

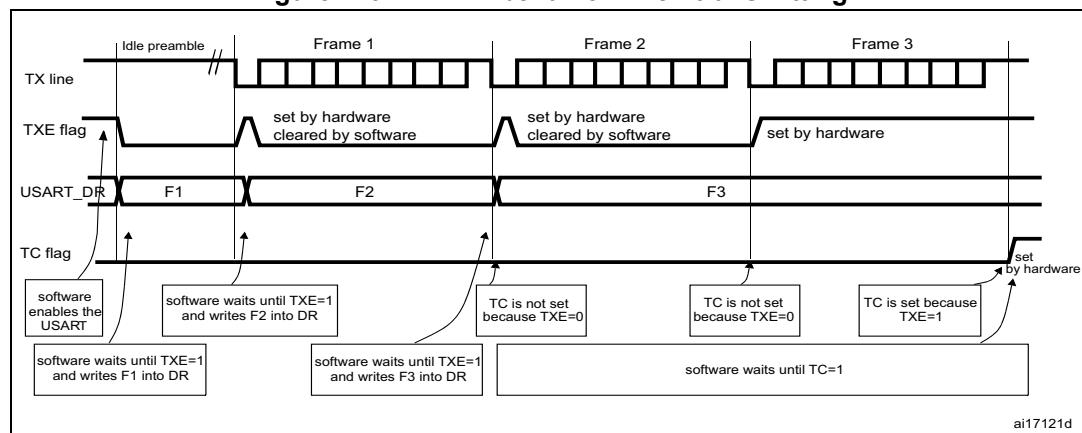
When no transmission is taking place, a write instruction to the UART\_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

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 UART\_CR2 register. After writing the last data in the UART\_DR register, it is mandatory to wait until TC is set to '1' before entering Halt mode or disabling the UART (see [Figure 116: TC/TXE behavior when transmitting](#)).

The following software sequence is required to clear the TC bit:

1. Read from the UART\_SR register
2. Write to the UART\_DR register

**Figure 116. TC/TXE behavior when transmitting**



1. This example assumes that several other transmissions occur after TE has been set.

Note:

*The TC bit can also be cleared by writing a '0' to it. This clear sequence is recommended only for multibuffer communication.*

### Break character

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see [Figure 114](#)).

If the SBK bit is set to '1' a break character is sent on the UART\_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 UART 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 UART 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 UART to send an idle frame before the first data frame.

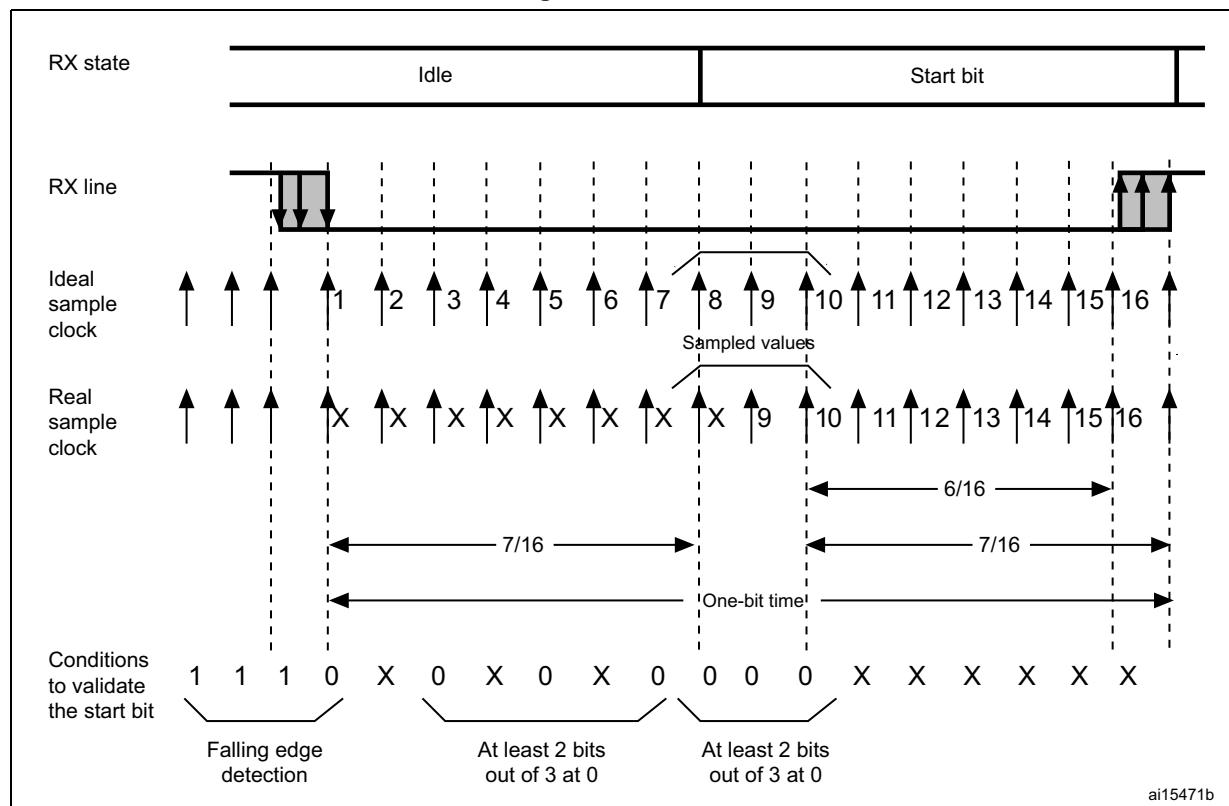
## 22.3.3 Receiver

The UART 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 UART\_CR1 register.

### Start bit detection

In the UART, 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 117](#).

Figure 117. 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 3<sup>rd</sup>, 5<sup>th</sup> and 7<sup>th</sup> bits or sampling on the 8<sup>th</sup>, 9<sup>th</sup> and 10<sup>th</sup> 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 8<sup>th</sup>, 9<sup>th</sup>, and 10<sup>th</sup> bits).

### Character reception

During an UART reception, data shifts in least significant bit first through the UART\_RX pin. In this mode, the UART\_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 UART\_CR1 to define the word length.
2. Program the number of stop bits in UART\_CR3.
3. Select the desired baud rate by programming the baud rate registers in the following order:
  - a) UART\_BRR2
  - b) UART\_BRR1
4. Set the REN bit UART\_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 UART\_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 UART handles it as a framing error.

### Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the ILIEN bit is set.

### Overrun error

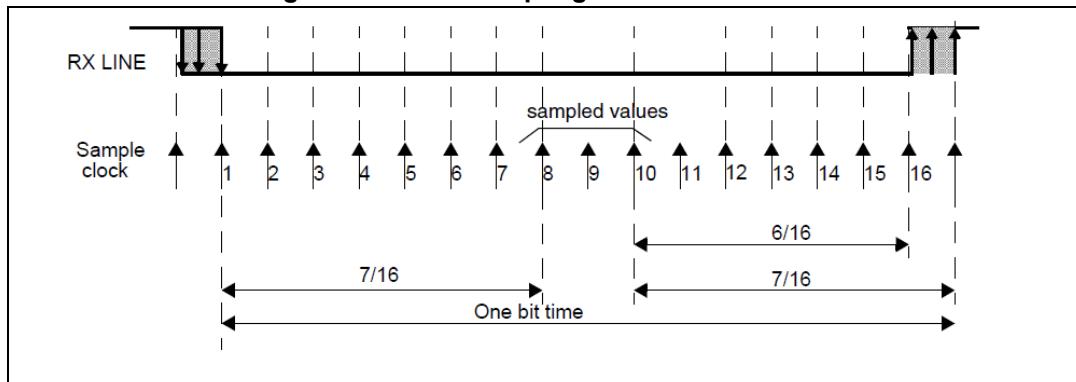
An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

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 UART\_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 UART\_SR register followed by a UART\_DR register read operation.

### Noise error

Over-sampling techniques are used for data recovery by discriminating between valid incoming data and noise.

**Figure 118. Data sampling for noise detection**

*Note:* The sample clock frequency is 16x baud rate.

**Table 53. 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 UART\_DR register.

This bit rises at the same time as the RXNE bit which generates an interrupt. The NF bit is reset by a UART\_SR register read operation followed by a UART\_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 UART\_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 UART\_SR register read operation followed by a UART\_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 in normal mode, 1 in IrDA mode and 1.5 in Smartcard mode.

1. 1 Stop Bit: Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
2. 1.5 Stop Bits (Smartcard mode only): Sampling for 1.5 stop bits is done on the 16th, 17th and 18th samples. An NACK signal received from the Smartcard forces the data signal low during the sampling, flagged as a framing error. Then, the FE flag is set with the RXNE at the end of the 1.5 stop bit.
3. 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.

### 22.3.4 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 `UART_DIV` according to the following formula:

$$\text{Tx/ Rx baud rate} = \frac{f_{\text{MASTER}}}{\text{UART\_DIV}}$$

The `UART_DIV` baud rate divider is an unsigned integer, coded in the `BRR1` and `BRR2` registers as shown in [Figure 119](#).

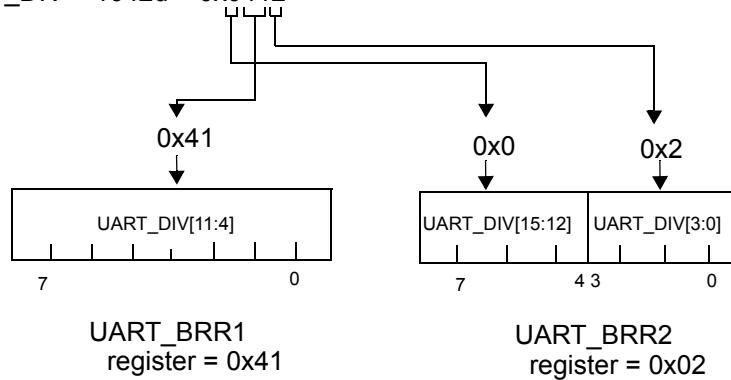
Refer to [Table 54](#) for typical baud rate programming examples.

**Figure 119. How to code `UART_DIV` in the `BRR` registers**

Example: To obtain 9600 baud with  $f_{\text{MASTER}} = 10 \text{ MHz}$ .

$$\text{UART\_DIV} = 10\,000\,000 / 9600$$

$$\text{UART\_DIV} = 1042d = 0x0412 \quad \text{See } \text{Table 54}.$$



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:

*`UART_DIV` must be greater than or equal to 16d.*

**Table 54. Baud rate programming and error calculation**

Baud rate	$f_{MASTER} = 10 \text{ MHz}$					$f_{MASTER} = 16 \text{ MHz}$				
	in kbps	Actual	% Error <sup>(1)</sup>	UART_DIV	BRR1	BRR2	Actual	% Error <sup>(1)</sup>	UART_DIV	BRR1
2.4	2.399	-0.008%	0x1047	0x04	0x17	2.399	-0.005%	0x1A0B	0xA0	0x1B
9.6	9.596	-0.032%	0x0412	0x41	0x02	9.598	-0.020%	0x0693	0x68h	0x03
19.2	19.193	-0.032%	0x0209	0x20	0x09	19.208	-0.040%	0x0341	0x34	0x01
57.6	57.471	-0.224%	0x00AE	0x0A	0x0E	57.554	-0.080%	0x0116	0x11	0x06
115.2	114.942	-0.224%	0x0057	0x05	0x07	115.108	-0.080%	0x008B	0x08	0x0B
230.4	232.558	-0.937%	0x002B	0x02	0x0B	231.884	-0.644%	0x0045	0x04	0x05
460.8	454.545	-1.358%	0x0016	0x01	0x06	457.143	-0.794%	0x0023	0x02	0x03
921.6	NA	NA	NA	NA	NA	941.176	-2.124%	0x11	0x01	0x01

1. Error % = (Calculated - Desired) Baud Rate / Desired Baud Rate

**Note:** The lower the  $f_{MASTER}$  frequency, the lower will be the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with this data.

### 22.3.5 Clock deviation tolerance of the UART receiver

The USART's asynchronous receiver works correctly only if the total clock system deviation is less 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)

$$\text{DTRA} + \text{DQUANT} + \text{DREC} + \text{DTCL} < \text{UART receiver's tolerance}$$

The UART 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 UART\_CR1 register
- Use of fractional baud rate or not

**Table 55. UART receiver tolerance when UART\_DIV[3:0] is zero**

M bit	NF is an error	NF is don't care
0	3.75%	4.375%
1	3.41%	3.97%

**Table 56. UART receiver's tolerance when UART\_DIV[3:0] is different from zero**

M bit	NF is an error	NF is don't care
0	3.33%	3.88%
1	3.03%	3.53%

**Note:**

The values specified in [Table 55](#) and [Table 56](#) 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).

### 22.3.6 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 UART\_CR1 register. Depending on the frame length defined by the M bit, the possible UART frame formats are as listed in [Table 57](#).

**Table 57. Frame format**

M bit	PCEN bit	UART 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 UART\_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 UART\_CR1 = 1).

**Transmission:** If the PCEN bit is set in UART\_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 UART\_SR register and an interrupt is generated if the PIEN bit is set in the UART\_CR1 register.

### 22.3.7 Multi-processor communication

It is possible to perform multi-processor communication with the UART (several UARTs connected in a network). For example, one of the UARTs can be the master, its TX output is connected to the RX input of the other UART. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multi-processor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant UART 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 UART\_CR2 register is set to 1. RWU can be controlled automatically by hardware or written by the software under certain conditions.

The UART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the UART\_CR1 register:

- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

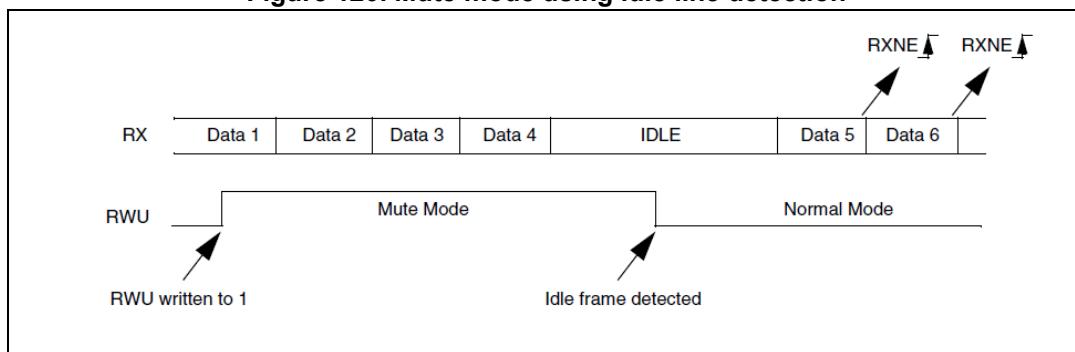
#### Idle line detection (WAKE = 0)

The UART 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 UART\_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 120](#).

**Figure 120. 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 UART\_CR4 register.

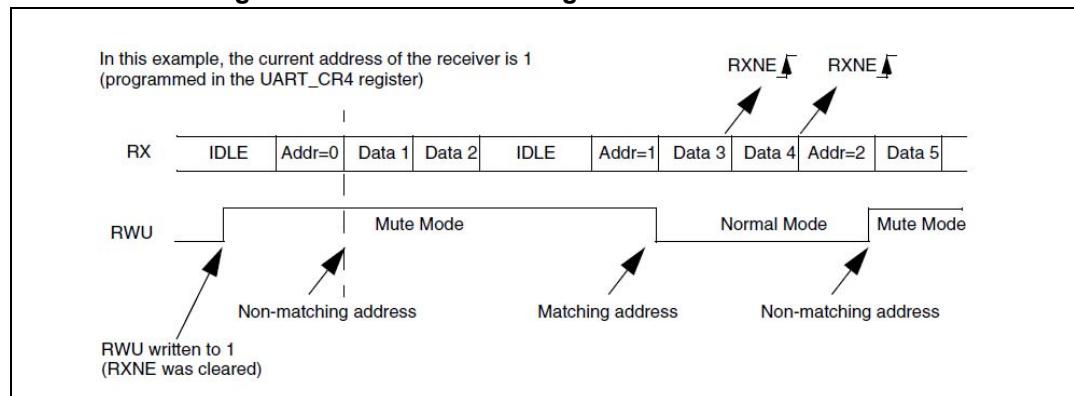
The UART enters mute mode when an address character is received which does not match its programmed address. The RXNE flag is not set for this address byte and no interrupt request is issued as the UART would have entered mute mode.

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 UART\_SR register). Otherwise the write attempt is ignored.

An example of mute mode behavior using address mark detection is given in [Figure 121](#).

**Figure 121. 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 | 7-bit data | ADD | PB | STB

where:

SB = Start Bit

STB = Stop Bit

ADD = Address bit

PB = Parity Bit

### 22.3.8 LIN (local interconnection network) mode

The UART supports LIN break and delimiter generation in LIN master mode.

Refer to [Section 22.4.1: Master mode on page 348](#) for details. LIN slave mode is supported by the UART2, UART3 and UART4 only, not by UART1.

LIN mode is selected by setting the LINEN bit in the UART\_CR3 register. In LIN mode, the following bits must be kept cleared:

- CLKEN, STOP[1:0] in the UART\_CR3 register
- SCEN, HDSEL and IREN in the UART\_CR5 register

### 22.3.9 UART synchronous communication

The UART transmitter allows the user to control bidirectional synchronous serial communications in master mode.

In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the UART\_CR3 register
- SCEN, HDSEL and IREN bits in the UART\_CR5 register

*Note:* *This feature is only available in UART1, UART2 and UART4.*

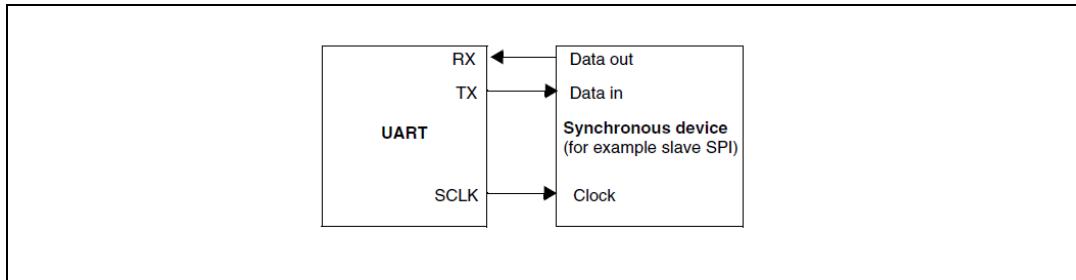
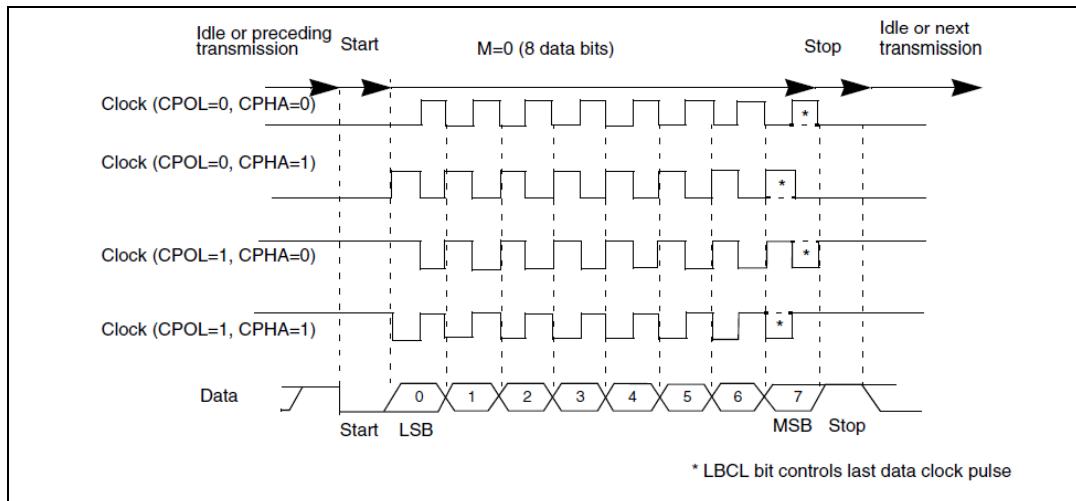
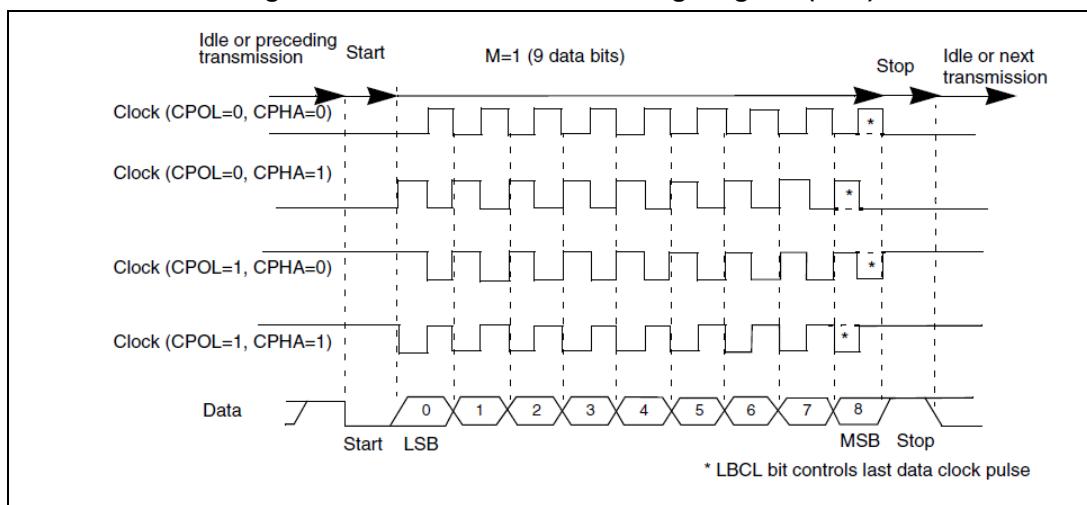
The UART\_CK pin is the output of the UART transmitter clock. No clock pulses are sent to the UART\_CK pin during start bit and stop bit. Depending on the state of the LBCL bit in the UART\_CR3 register clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the UART\_CR3 register allows the user to select the clock polarity, and the CPHA bit in the UART\_CR3 register allows the user to select the phase of the external clock (see [Figure 122](#), [Figure 123](#) & [Figure 124](#)).

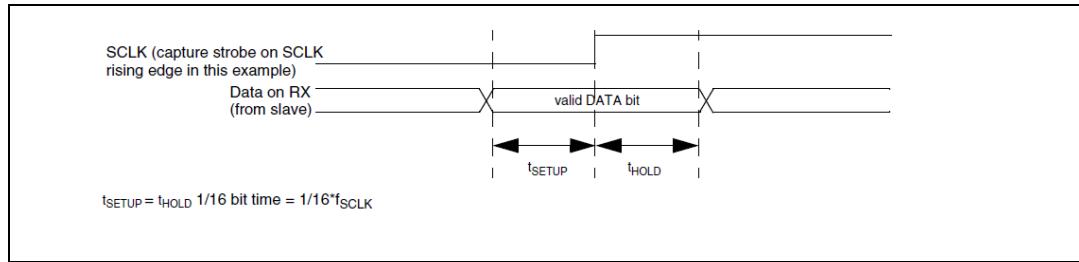
During idle and break frames, the external CK clock is not activated.

In synchronous mode, the UART receiver works differently compared to asynchronous mode. If RE=1, the data is sampled on SCLK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time (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).

- Note:*
- 1 *The UART\_CK pin works in conjunction with the TX pin. When the UART transmitter is disabled (TEN and REN= 0), the UART\_CK and UART\_TX pins go into high impedance state.*
  - 2 *The LBCL, CPOL and CPHA bits in UART\_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 TE and RE are set in the same instruction in order to minimize the setup and the hold time of the receiver.*
  - 4 *The UART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).*
  - 5 *The data given in this section apply only when the UART\_DIV[3:0] bits in the UART\_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 122. UART example of synchronous transmission****Figure 123. UART data clock timing diagram (M=0)****Figure 124. UART data clock timing diagram (M=1)**

**Figure 125. RX data setup/hold time**

**Note:** The function of `UART_CK` is different in Smartcard mode. Refer to Section 22.3.11: [Smartcard](#) for more details.

### 22.3.10 Single wire half duplex communication

The UART can be configured to follow a single wire half duplex protocol. Single-wire half-duplex mode is selected by setting the HDSEL bit in the `UART_CR5` register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the `UART_CR3` register
- SCEN and IREN bits in the `UART_CR5` register

**Note:** This feature is only available in `UART1` and `UART4`.

As soon as HDSEL is set:

- `UART_RX` is no longer used
- `UART_TX` is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception. This means that the I/O must be configured so that `UART_TX` is configured as floating input (or output high open-drain) when not driven by the UART.

Apart from this, the communications are similar to what is done in normal UART mode. The conflicts on the line must be managed by the software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continue to occur as soon as a data is written in the data register while the TEN bit is set.

### 22.3.11 Smartcard

Smartcard mode is selected by setting the SCEN bit in the `UART_CR5` register. In smartcard mode, the following bits must be kept cleared:

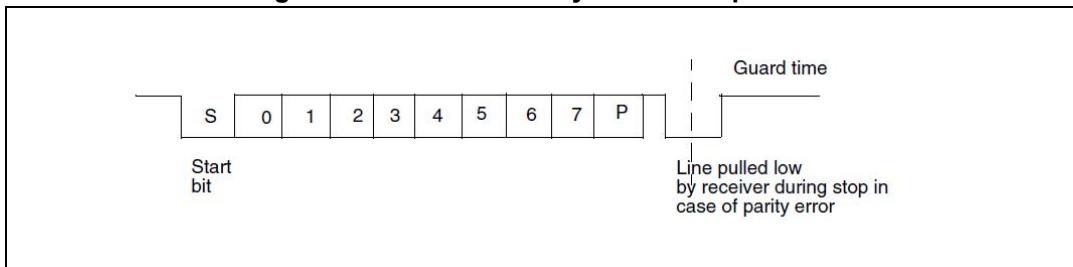
- LINEN bit in the `UART_CR3` register,
- HDSEL and IREN bits in the `UART_CR5` register.

Moreover, the CKEN bit may be set in order to provide a clock to the smartcard.

**Note:** This feature is only available in `UART1`, `UART2` and `UART4`.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO7816-3 standard. The UART should be configured as eight bits plus parity and 1.5 stop bits. With Smartcard mode enabled (which can be done by setting the SCEN bit in the `UART_CR5`) the UART can communicate with an asynchronous Smartcard.

Figure 126. ISO 7816-3 asynchronous protocol



When connected to a smartcard, the UART\_TX output drives a bidirectional line that is also driven by the smartcard.

Smartcard is a single wire half duplex communication protocol.

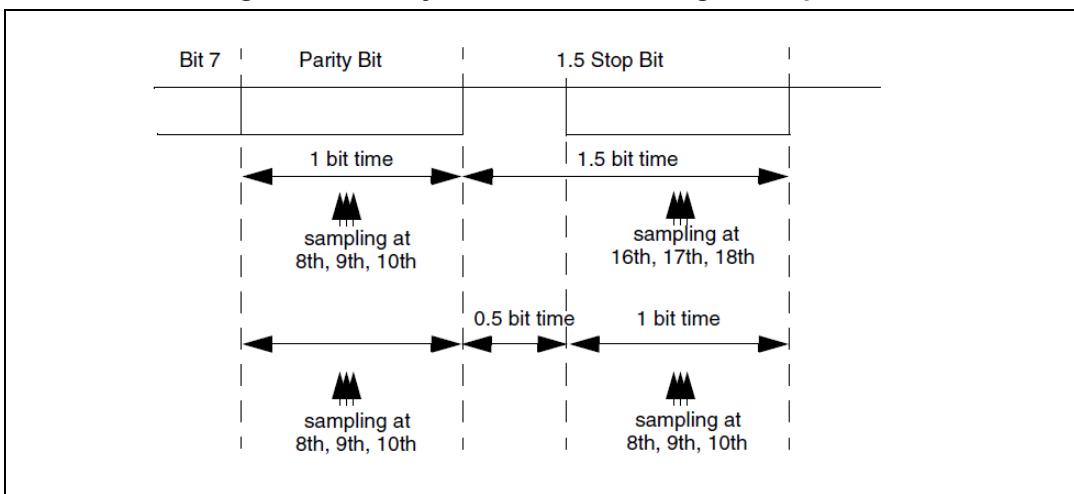
- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register will start shifting on the next baud clock edge. In Smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- If a parity error is detected during reception of a frame programmed with a 1.5 stop bit period, the transmit line is pulled low for a baud clock period after 1/2 baud clock period. This is to indicate to the Smartcard that the data transmitted to the UART has not been correctly received. This NACK signal (pulling transmit line low for 1 baud clock) will cause a framing error on the transmitter side (configured with 1.5 stop bits). The application can handle re-sending of data according to the protocol. A parity error is 'NACK'ed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted.
- The TE bit must be set to enable:
  - Data transmission
  - Transmission of acknowledgements in case of parity error.
 Software must manage the timing of data transmission to avoid conflicts on the data line when it writes new data in the data register.
- The RE bit must be set to enable:
  - Data reception (sent by the Smartcard as well as by the UART),
  - Detection of acknowledgements in case of parity error.
- The assertion of the TC flag can be delayed by programming the Guard Time register. In normal operation, TC is asserted when the transmit shift register is empty and no further transmit requests are outstanding. In Smartcard mode an empty transmit shift register triggers the guard time counter to count up to the programmed value in the Guard Time register. TC is forced low during this time. When the guard time counter reaches the programmed value TC is asserted high.
- The de-assertion of TC flag is unaffected by Smartcard mode.
- If a framing error is detected on the transmitter end (due to a NACK from the receiver), the NACK will not be detected as a start bit by the receive block of the transmitter. According to the ISO protocol, the duration of the received NACK can be 1 or 2 baud clock periods.
- On the receiver side, if a parity error is detected and a NACK is transmitted the receiver will not detect the NACK as a start bit.

- The output enable signal for the Smartcard I/O enables driving into a bidirectional line which is also driven by the Smartcard. This signal is active while transmitting the start and data bits and transmitting NACK. While transmitting the stop bits this signal is disabled, so that the UART weakly drives a '1' on the bidirectional line.

- Note:**
- 1 A break character is not significant in Smartcard mode. A 00h data with a framing error will be treated as data and not as a break.
  - 2 No IDLE frame is transmitted when toggling the TEN bit. The IDLE frame (as defined for the other configurations) is not defined by the ISO protocol.

[Figure 127](#) details how the NACK signal is sampled by the UART. In this example the UART is transmitting a data and is configured with 1.5 stop bits. The receiver part of the UART is enabled in order to check the integrity of the data and the NACK signal.

**Figure 127. Parity error detection using 1.5 stop bits**



The UART can provide a clock to the smartcard through the UART\_CK output. In smartcard mode, UART\_CK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler. The division ratio is configured in the prescaler register UART\_PSCR. UART\_CK frequency can be programmed from  $f_{MASTER}/2$  to  $f_{MASTER}/62$ , where  $f_{MASTER}$  is the peripheral input clock.

### 22.3.12 IrDA SIR ENDEC block

IrDA mode is selected by setting the IREN bit in the UART\_CR5 register. The STOP bits in the UART\_CR3 register must be configured to "1 stop bit". In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CKEN bits in the UART\_CR3 register,
- SCEN and HDSEL bits in the UART\_CR5 register.

**Note:** This feature is only available in UART1, UART2 and UART4.

The IrDA SIR physical layer specifies use of a Return to Zero, Inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse (see [Figure 128](#)).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from the UART. The output pulse stream is transmitted to an external output driver and infrared LED. The UART supports only bit rates up to 115.2 kbps for the SIR ENDEC. In normal mode the transmitted pulse width is specified as 3/16 of a bit period.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to UART. The decoder input is normally HIGH (marking state) in the idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (i.e. the UART is sending data to the IrDA encoder), any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy (UART is receiving decoded data from the UART), data on the TX from the UART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.
- A '0' is transmitted as a high pulse and a '1' is transmitted as a '0'. The width of the pulse is specified as 3/16th of the selected bit period in normal mode (see [Figure 129](#)).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for the UART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when idle.
- The IrDA specification requires the acceptance of pulses greater than 1.41 us. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in `UART_PSCR`). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods will be accepted as a pulse. The IrDA encoder/decoder doesn't work when `PSC=0`.
- The receiver can communicate with a low-power transmitter.
- In IrDA mode, the STOP bits in the `UART_CR2` register must be configured to "1 stop bit".

### **IrDA low-power mode**

The IrDA can be used either in normal mode or in Low Power mode. The Low Power mode is selected by setting the `IRLP` bit in `UART_CR5` register.

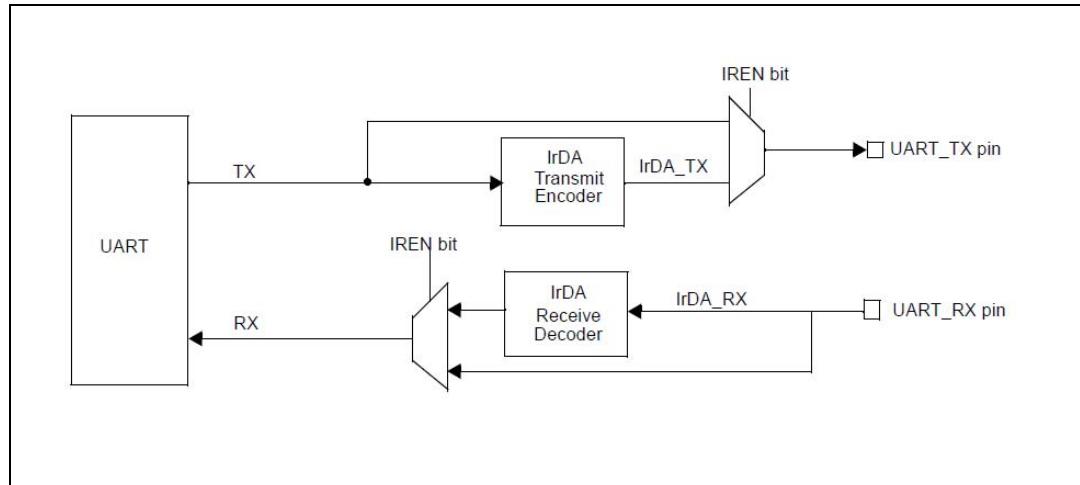
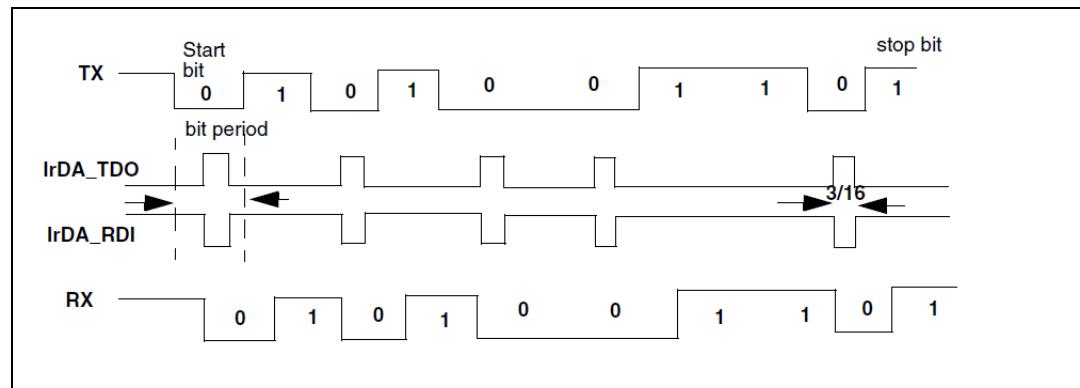
#### **Transmitter:**

In low-power mode the pulse width is not maintained at 3/16 of the bit period. Instead, the width of the pulse is 3 times the low-power baud rate which can be a minimum of 1.42 MHz. Generally this value is 1.8432 MHz ( $1.42\text{ MHz} < \text{PSC} < 2.12\text{ MHz}$ ). A low-power mode programmable divisor divides the system clock to achieve this value.

#### **Receiver:**

Receiving in low-power mode is similar to receiving in normal mode. For glitch detection the UART should discard pulses of duration shorter than 1/PSC. A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in `UART_PSCR`).

- Note:**
- 1 *A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.*
  - 2 *The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).*

**Figure 128. IrDA SIR ENDEC- block diagram****Figure 129. IrDA data modulation (3/16) - normal mode**

## 22.4 LIN mode functional description

In LIN mode, 8-bit data format with 1 stop bit is required in accordance with the LIN standard.

To configure these settings, clear the M bit in UART\_CR1 register and clear the STOP[1:0] bits in the UART\_CR3 register.

### 22.4.1 Master mode

#### UART initialization

Procedure:

1. Select the desired baudrate by programming the UART\_BRR2 and UART\_BRR1 registers.
2. Enable LIN mode by setting the LINEN bit in the UART\_CR3 register.
3. Enable the transmitter and receiver by setting the TEN and REN bits in the UART\_CR2 register.

#### LIN header transmission

According to the LIN protocol, any communication on the LIN bus is triggered by the Master sending a Header, followed by the response. The Header is transmitted by the Master Task (master node) while the data are transmitted by the Slave task of a node (master node or one of the slave nodes).

Procedure without error monitoring:

1. Request Break + Delimiter transmission (13 dominant bits and 1 recessive bit) by setting the SBK bit in the UART\_CR2 register.
2. Request Synch Field transmission by writing 0x55 in the UART\_DR register.
3. Wait for the TC flag in the UART\_SR register.
4. Request Identifier Field transmission by writing the protected identifier value in the UART\_DR register.
5. Wait for the TC flag in the UART\_SR register.

Procedure with error monitoring:

1. Request Break + Delimiter transmission (13 dominant bits and 1 recessive bit) by setting the SBK bit in the UART\_CR2 register;
2. Wait for the LBDF flag in the UART\_CR4 register.
3. Request Synch Field transmission by writing 0x55 into UART\_DR register.
4. Wait for the RXNE flag in the UART\_SR register and read back the UART\_DR register.
5. Request Identifier Field transmission by writing the protected identifier value in the UART\_DR register.
6. Wait for the RXNE flag in the UART\_SR register and read back the UART\_DR register.

The LBDF flag is set only if a valid Break + Delimiter has been received back on the UART\_RX pin.

## LIN break and delimiter detection

The UART features a break detection circuit which is totally independent from the normal UART receiver. A break can be detected whenever it occurs, during idle state or during a frame.

When the receiver is enabled (REN=1 in UART\_CR2), the circuit looks at the UART\_RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 bits (when the LBDL = 0 in UART\_CR4) or 11 bits (when LBDL=1 in UART\_CR4) are detected as '0', and are followed by a delimiter character, the LBDF flag is set in UART\_CR4. If the LBDIEN bit=1, an interrupt is generated.

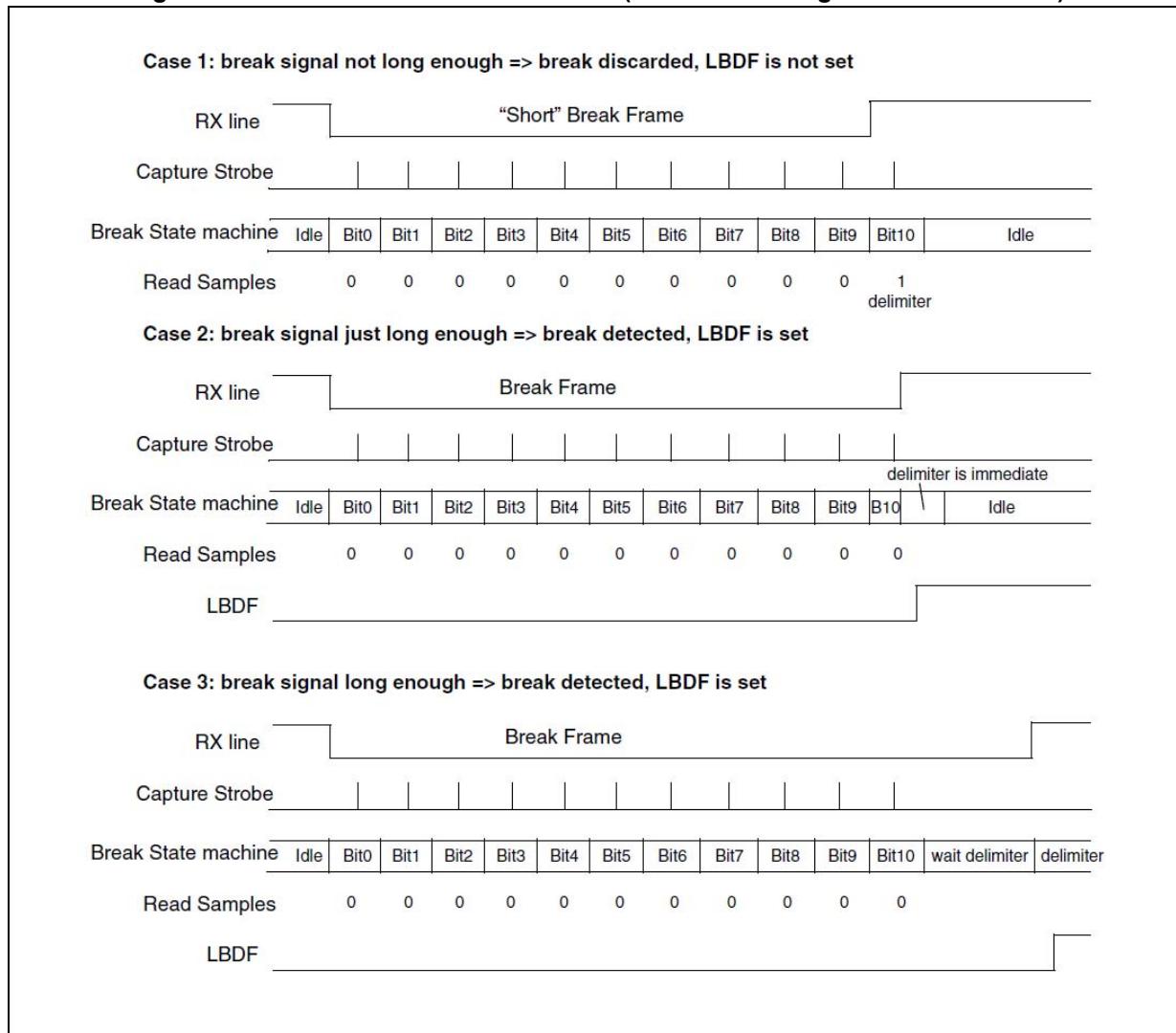
If a '1' is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again. If LIN mode is disabled (LINEN=0), the receiver continues working as a normal UART, without taking into account the break detection.

If LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at '0', which will be the case for any break frame), the receiver stops until the break detection circuit receives either a '1', if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown in [Figure 130: Break detection in LIN mode \(11-bit break length - LBDL bit is set\) on page 350](#).

The LBDF flag is used in master mode, in slave mode the LHDF flag is used instead.

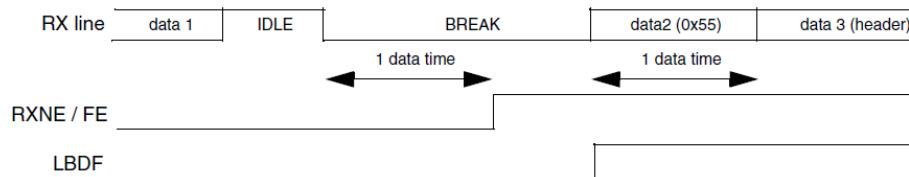
Examples of break frames are given on [Figure 131: Break detection in LIN mode vs framing error detection on page 351](#).

**Figure 130. Break detection in LIN mode (11-bit break length - LBDL bit is set)**

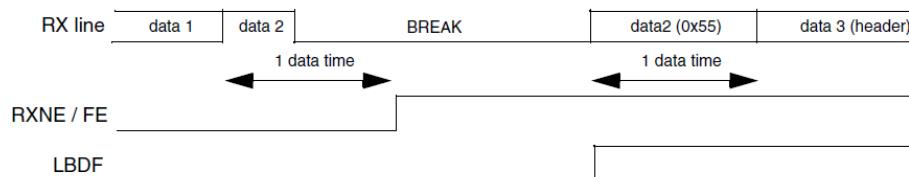
**Figure 131. Break detection in LIN mode vs framing error detection**

In these examples, we suppose that LBDL=1 (11-bit break length), M=0 (8-bit data)

**Case 1: break occurring after an Idle**



**Case 2: break occurring while a data is being received**



### Response transmission (master is the publisher of the response)

The response is composed of bytes with a standard UART format: 8-bit data, 1 stop bit, no parity.

In order to send n data bytes, the application must repeat the following sequence n times:

1. Write data in UART\_DR register
2. Wait for RXNE flag in UART\_SR register
3. Check for readback value by reading the UART\_DR register

### Response reception (master is the subscriber of the response)

In order to receive n data bytes, the application must repeat following sequence n times:

1. Wait for the RXNE flag in the UART\_SR register
2. Read the UART\_DR register

### Discard response (slave to slave communication)

In case of slave to slave communication and if the master does not need to check errors in the response, the application can ignore the RXNE flag till the next frame slot. The RXNE and OR flags should be cleared before starting the next Break transmission.

**Note:**

*Receiving back a Break will also set the RXNE and FE flags before setting the LBDF flag. Therefore, if the RX interrupt is used, it's better to disable it (by clearing the RIEN bit in the UART\_CR2 register) before sending the Break, to avoid an additional interrupt. In case of slave to slave communication, RIEN bit can be cleared once the header has been transmitted.*

## 22.4.2 Slave mode with automatic resynchronization disabled

*Note:* This feature is only available in UART2, UART3 and UART4.

### UART initialization

Procedure:

1. Select the desired baudrate by programming UART\_BRR2 and UART\_BRR1 registers,
2. Enable transmitter and receiver by setting TEN and REN bits in UART\_CR2 register,
3. Enable LSLV bit in UART\_CR6 register,
4. Enable LIN mode by setting LINEN bit in UART\_CR3 register,

### LIN Header reception

According to the LIN protocol, a slave node must wait for a valid header, coming from the master node. Then application has to take following action, depending on the header Identifier value:

- Receive the response
- Transmit the response
- Ignore the response and wait for next header

When a LIN Header is received:

- The LHDF flag in the UART\_CR6 register indicates that a LIN Header has been detected.
- An interrupt is generated if the LHDIE bit in the UART\_CR6 register is set.
- The LIN Identifier is available in the UART\_DR register.

*Note:* It is recommended to put UART in mute mode by setting RWU bit. This mode allows detection of headers only and prevents the reception of any other characters.

Setting the PCEN bit in the UART\_CR2 register while LIN is in slave mode enables the Identifier parity check. The PE flag in the UART\_CR6 register is set together with the LHDF flag in the UART\_CR6 register if the Identifier parity is not correct.

### Response transmission (slave is the publisher of the response)

In order to send n data bytes, the application must repeat following sequence n times:

1. Write data in the UART\_DR register
2. Wait for the RXNE flag in the UART\_SR register
3. Check for readback value by reading the UART\_DR register

Once response transmission is completed, software can set the RWU bit.

### Response reception (slave is the subscriber of the response)

In order to receive n data bytes, the application must repeat following sequence n times:

1. Wait for the RXNE flag in the UART\_SR register
2. Read the UART\_DR register

Once response reception is completed, software can set the RWU bit.

### Discard response

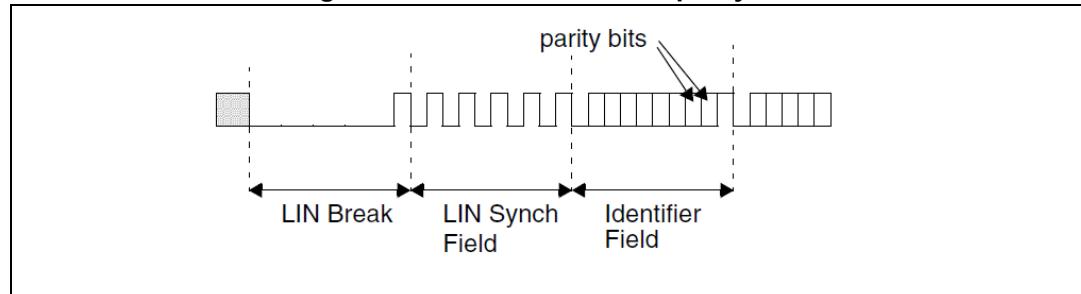
Software can set the RWU bit immediately.

#### LIN Slave parity

In LIN Slave mode (LINEN and LSLV bits are set) LIN parity checking can be enabled by setting the PCEN bit. An interrupt is generated if an ID parity error occurs (PE bit rises) and the PIEN bit is set.

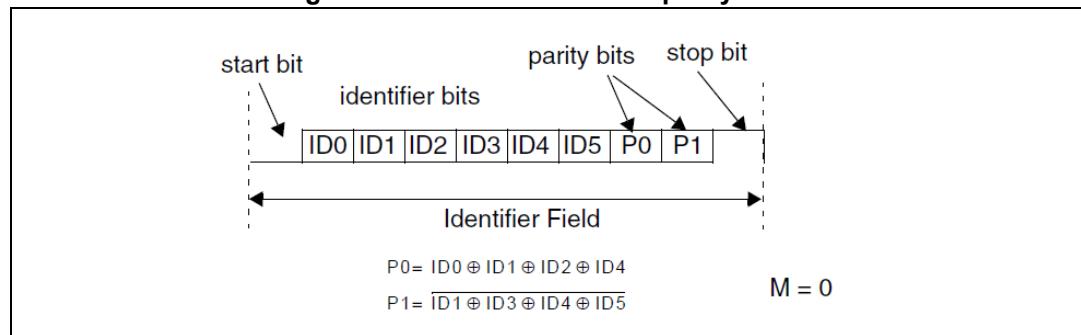
In this case, the parity bits of the LIN Identifier Field are checked. The identifier character is recognized as the third received character after a break character (included):

**Figure 132. LIN identifier field parity bits**



The bits involved are the two MSB positions (7th and 8th bits) of the identifier character. The check is performed as specified by the LIN specification:

**Figure 133. LIN identifier field parity check**



### LIN header error detection

The LIN Header Error Flag indicates that an invalid LIN Header has been detected.

When a LIN Header Error occurs:

- The LHE flag is set
- An interrupt is generated if the RIEN bit in the UART\_CR2 register is set.

The LHE bit is reset by an access to the UART\_SR register followed by a read of the UART\_DR register.

LHE is set if one of the following conditions occurs:

- Break Delimiter is too short
- Synch Field is different from 55h
- Framing error in Synch Field or Identifier Field
- A LIN header reception time-out

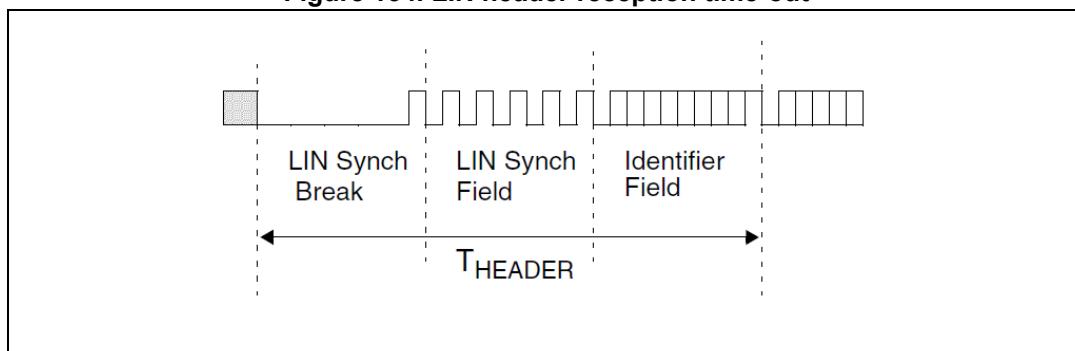
**Note:** If a LIN header error occurs, the LSF bit in the UART\_CR6 register must be cleared by software

### LIN header time-out error

The UART automatically monitors the THEADER\_MAX condition given by the LIN protocol.

If the entire Header (up to and including the STOP bit of the LIN Identifier Field) is not received within the maximum time limit of 57 bit times then a LIN Header Error is signaled and the LHE bit is set in the UART\_SR register.

**Figure 134. LIN header reception time-out**



The time-out counter is enabled at each break detection. It is stopped in the following conditions:

- A LIN Identifier Field has been received
- An LHE error occurred (other than a time-out error).
- A software reset of LSF bit (transition from high to low) occurred during the analysis of the LIN Sync Field

If LHE bit is set due to this error during the LIN Sync Field (if LASE bit = 1) then the UART goes into a blocked state (the LSF bit is set).

If LHE bit is set due to this error during Fields other than LIN Sync Field or if LASE bit is reset then the current received Header is discarded and the UART searches for a new Break Field.

#### Note on LIN Header time-out limit

According to the LIN specification, the maximum length of a LIN Header which does not cause a time-out is equal to:

$$1.4 * (34 + 1) = 49 \text{ TBIT\_MASTER}.$$

TBIT\_MASTER refers to the master baud rate.

When checking this time-out, the slave node is desynchronized for the reception of the LIN Break and Synch fields. Consequently, a margin must be allowed, taking into account the worst case: This occurs when the LIN identifier lasts exactly 10 TBIT\_MASTER periods. In this case, the LIN Break and Synch fields last  $49 - 10 = 39$  TBIT\_MASTER periods.

Assuming the slave measures these first 39 bits with a desynchronized clock of 15.5%. This leads to a maximum allowed Header Length of:

$$\begin{aligned} 39 \times (1/0.845) \text{ TBIT\_MASTER} + 10 \text{ TBIT\_MASTER} \\ = 56.15 \text{ TBIT\_SLAVE} \end{aligned}$$

A margin is provided so that the time-out occurs when the header length is greater than 57 TBIT\_SLAVE periods. If it is less than or equal to 57 TBIT\_SLAVE periods, then no time-out occurs.

### Mute mode and errors

In mute mode, if an LHE error occurs during the analysis of the LIN Synch Field or if a LIN Header Time-out occurs then the LHE bit is set but it does not wake up from mute mode. In this case, the current header analysis is discarded. If needed, the software has to reset the LSF bit. Then the UART searches for a new LIN header.

In mute mode, if a framing error occurs on a data (which is not a break), it is discarded and the FE bit is not set.

Any LIN header which respects the following conditions causes a wake-up from mute mode:

- A valid LIN Break and Delimiter
- A valid LIN Synch Field (without deviation error)
- A LIN Identifier Field without framing error. Note that a LIN parity error on the LIN Identifier Field does not prevent wake-up from mute mode.
- No LIN Header Time-out should occur during Header reception.

#### 22.4.3 Slave mode with automatic resynchronization enabled

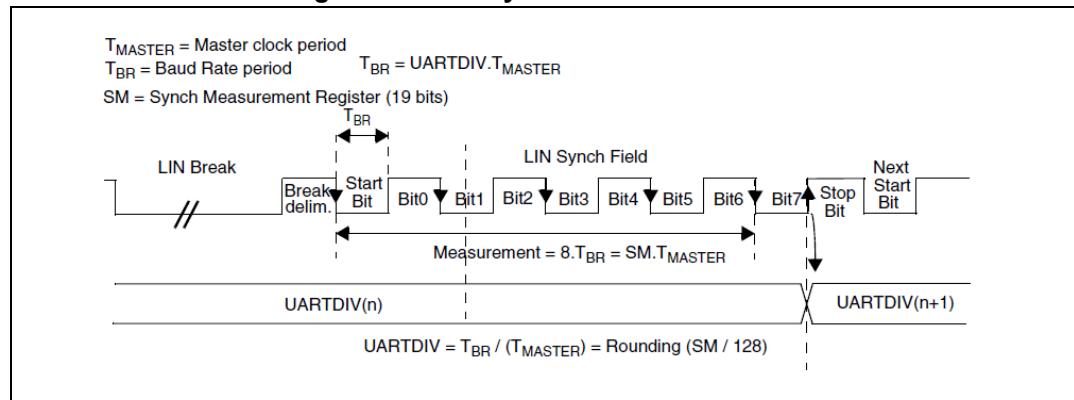
This mode is similar to slave mode as described in [Section 22.4.2: Slave mode with automatic resynchronization disabled](#), with the addition of automatic resynchronization enabled by the LASE bit. In this mode UART adjusts the baudrate generator after each Synch Field reception.

*Note:* This feature is only available at UART2, UART3 and UART4.

### Automatic resynchronization

When automatic resynchronization is enabled, after each LIN Break, the time duration between 5 falling edges on RDI is sampled on  $f_{MASTER}$  and the result of this measurement is stored in an internal 19-bit register called SM (not user accessible) (See [Figure 135](#)). Then the UARTDIV value (and its associated BRR1 and BRR2 registers) are automatically updated at the end of the fifth falling edge. During LIN Synch field measurement, the UART state machine is stopped and no data is transferred to the data register.

Figure 135. LIN synch field measurement



$\text{UARTDIV}$  is an unsigned integer, coded in the BRR1 and BRR2 registers as shown in [Figure 119](#).

If LASE bit = 1 then  $\text{UARTDIV}$  is automatically updated at the end of each LIN Sync Field.

Three registers are used internally to manage the auto-update of the LIN divider ( $\text{UARTDIV}$ ):

- $\text{UARTDIV\_NOM}$  (nominal value written by software at  $\text{UART\_BRR1}$  and  $\text{UART\_BRR2}$  addresses)
- $\text{UARTDIV\_MEAS}$  (results of the Field Synch measurement)
- $\text{UARTDIV}$  (used to generate the local baud rate)

The control and interactions of these registers are explained in [Figure 136](#) and [Figure 137](#). They depend on the LDUM bit setting (LIN Divider Update Method).

As explained in [Figure 136](#) and [Figure 137](#),  $\text{UARTDIV}$  can be updated by two concurrent actions: a transfer from  $\text{UARTDIV\_MEAS}$  at the end of the LIN Sync Field and a transfer from  $\text{UARTDIV\_NOM}$  due to a software write to BRR1. If both operations occur at the same time, the transfer from  $\text{UARTDIV\_NOM}$  has priority.

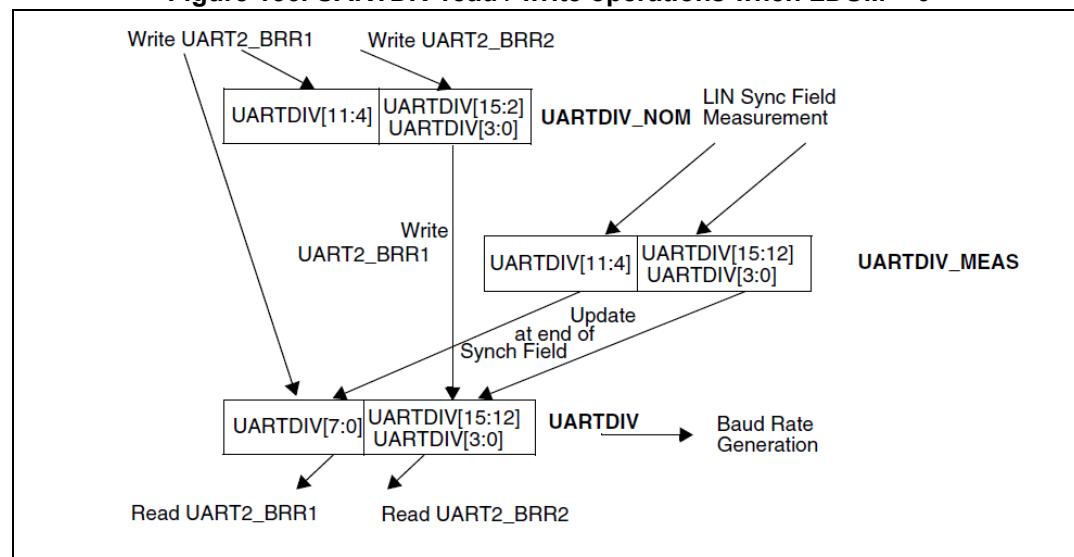
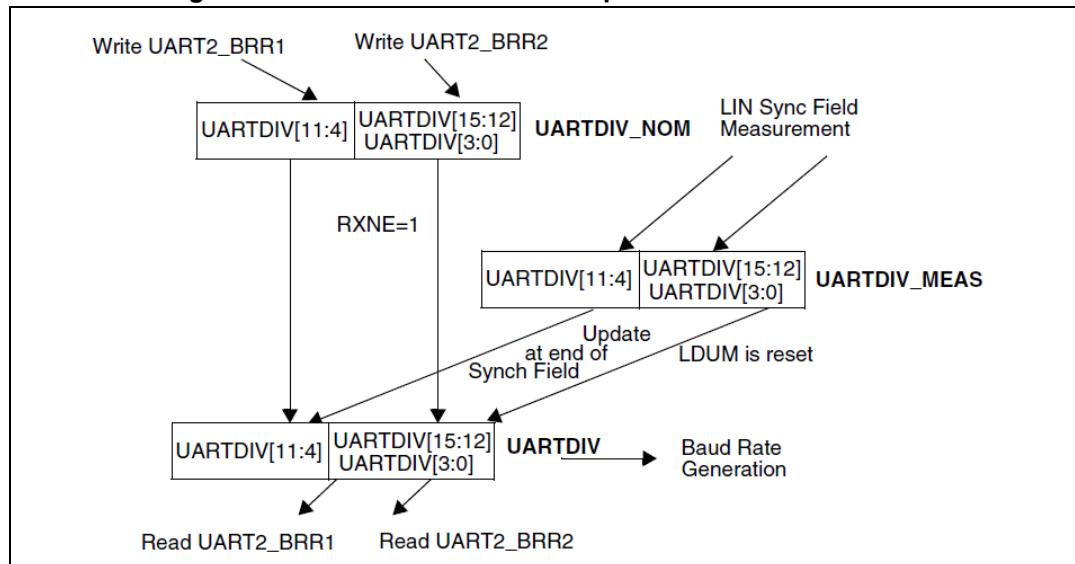
Figure 136.  $\text{UARTDIV}$  read / write operations when  $\text{LDUM} = 0$ 

Figure 137. UARTDIV read / write operations when LDUM = 1



#### Deviation error on the synch field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN Sync Field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the Synch Field.

- If  $D1 > 14.84\% LHE$  is set
- If  $D1 < 14.06\% LHE$  is not set
- If  $14.06\% < D1 < 14.84\% LHE$  can be either set or reset depending on the dephasing between the signal on **UART\_RX** pin and the  $f_{MASTER}$  clock

The second check is based on a measurement of time between each falling edge of the Synch Field

- If  $D2 > 18.75\% LHE$  is set
- If  $D2 < 15.62\% LHE$  is not set
- If  $15.62\% < D2 < 18.75\% LHE$  can be either set or reset depending on dephasing between the signal on **UART\_RX** pin and the  $f_{MASTER}$  clock

Note that the UART does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

**Note:** *Deviation checking is based on the current baudrate and not on the nominal one. Therefore, in order to guarantee correct deviation checking, the baudrate generator must reload the nominal value before each new Break reception. This nominal value is programmed by the application during initialization. To do this software must set the LDUM bit before checksum reception.*

*If LDUM bit is set, the next character reception will automatically reload the baudrate generator with nominal value.*

*You can also reload the nominal value by writing to BRR2 and BRR1. This second method is typically used when an error occurs during response transmission or reception.*

*If for any reason, the LDUM bit is set when UART is receiving a new Break and a Synch Field, this bit will be ignored and cleared. UART will adjust the baudrate generator with a value calculated from the synch field.*

### LIN header error detection

LHE is set if one of the following conditions occurs:

- Break Delimiter is too short
- Deviation error on the Synch Field is outside the LIN specification which allows up to +/- 14% of period deviation between the slave and master oscillators.
- Framing error in Synch Field or Identifier Field
- A LIN header reception time-out
- An overflow during the Synch Field Measurement, which leads to an overflow of the divider registers

### LIN header time-out error

The description in the section [LIN header time-out error on page 354](#) applies also when automatic resynchronization is enabled.

### UART clock tolerance when synchronized

When synchronization has been performed, following reception of a LIN Break, the UART has the same clock deviation tolerance as in UART mode, which is explained below:

During reception, each bit is oversampled 16 times. The mean of the 8th, 9th and 10th samples is considered as the bit value.

Consequently, the clock frequency should not vary more than 6/16 (37.5%) within one bit.

The sampling clock is resynchronized at each start bit, so that when receiving 10 bits (one start bit, 1 data byte, 1 stop bit), the clock deviation should not exceed 3.75%.

### UART clock tolerance when unsynchronized

When LIN slaves are unsynchronized (meaning no characters have been transmitted for a relatively long time), the maximum tolerated deviation of the UART clock is +/-14%.

If the deviation is within this range then the LIN Break is detected properly when a new reception occurs.

This is made possible by the fact that masters send 13 low bits for the LIN Break, which can be interpreted as 11 low bits (13 bits -14% = 11.18) by a "fast" slave and then considered as a LIN Break. According to the LIN specification, a LIN Break is valid when its duration is greater than  $t_{SBRKTS} = 10$ . This means that the LIN Break must last at least 11 low bits.

If the period desynchronization of the slave is +14% (slave too slow), the character "00h" which represents a sequence of 9 low bits must not be interpreted as a break character (9 bits + 14% = 10.26). Consequently, a valid LIN break must last at least 11 low bits.

### Clock deviation causes

The causes which contribute to the total deviation are:

- DTRA: Deviation due to transmitter error. Note: the transmitter can be either a master or a slave (in case of a slave listening to the response of another slave).
- DMEAS: Error due to the LIN Synch measurement performed by the receiver.
- DQUANT: Error due to the baud rate quantization of the receiver.
- DREC: Deviation of the local oscillator of the receiver: This deviation can occur during the reception of one complete LIN message assuming that the deviation has been compensated at the beginning of the message.
- DTCL: Deviation due to the transmission line (generally due to the transceivers)
- All the deviations of the system should be added and compared to the UART clock tolerance:
  - $DTRA + DMEAS + DQUANT + DREC + DTCL < 3.75\%$

### Error due to LIN synch measurement

The LIN Synch Field is measured over eight bit times.

This measurement is performed using a counter clocked by the CPU clock. The edge detections are performed using the CPU clock cycle.

This leads to a precision of 2 CPU clock cycles for the measurement which lasts 8\*UARTDIV clock cycles.

Consequently, this error (DMEAS) is equal to:

$$2 / (8 * \text{UARTDIVMIN})$$

UARTDIVMIN corresponds to the minimum LIN prescaler content, leading to the maximum baud rate, taking into account the maximum deviation of +/-14%.

### Error due to baud rate quantization

The baud rate can be adjusted in steps of 1 / (UARTDIV). The worst case occurs when the "real" baud rate is in the middle of the step.

This leads to a quantization error (DQUANT) equal to 1 / (2\*UARTDIVMIN).

### Impact of clock deviation on maximum baud rate

The choice of the nominal baud rate (UARTDIVNOM) will influence both the quantization error (DQUANT) and the measurement error (DMEAS). The worst case occurs for UARTDIVMIN.

Consequently, at a given CPU frequency, the maximum possible nominal baud rate (LPRMIN) should be chosen with respect to the maximum tolerated deviation given by the equation:

$$DTRA + 1 / (2 * \text{UARTDIVMIN}) + DREC + DTCL < 3.75\%$$

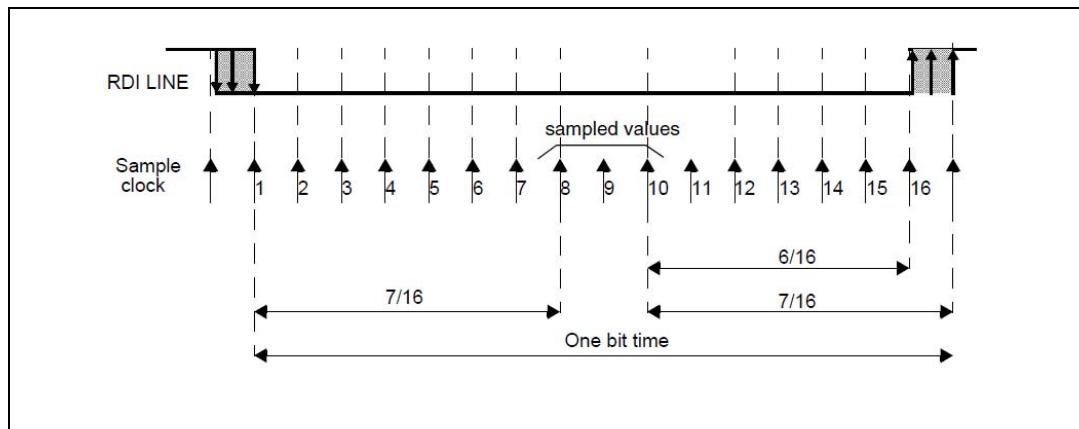
#### Example:

A nominal baud rate of 20 Kbits/s at TCPU = 125 ns (8 MHz) leads to UARTDIVNOM = 25d.

$$\text{UARTDIVMIN} = 25 - 0.15 * 25 = 21.25$$

$$\text{DQUANT} = 1 / (2 * \text{UARTDIVMIN}) = 0.0015\%$$

**Figure 138. Bit sampling in reception mode**



#### 22.4.4 LIN mode selection

**Table 58. LIN mode selection**

LINE	LSLV	LASE	Meaning
0	0	0	LIN mode disabled
			LIN Master Mode
1	1	0	LIN Slave Mode with Automatic resynchronization disabled
		1	LIN Slave Mode with Automatic resynchronization enabled

## 22.5 UART low power modes

**Table 59. UART interface behavior in low power modes**

Mode	Description
Wait	No effect on UART. UART interrupts cause the device to exit from Wait mode.
Halt	UART registers are frozen. In Halt mode, the UART stops transmitting/receiving until Halt mode is exited.

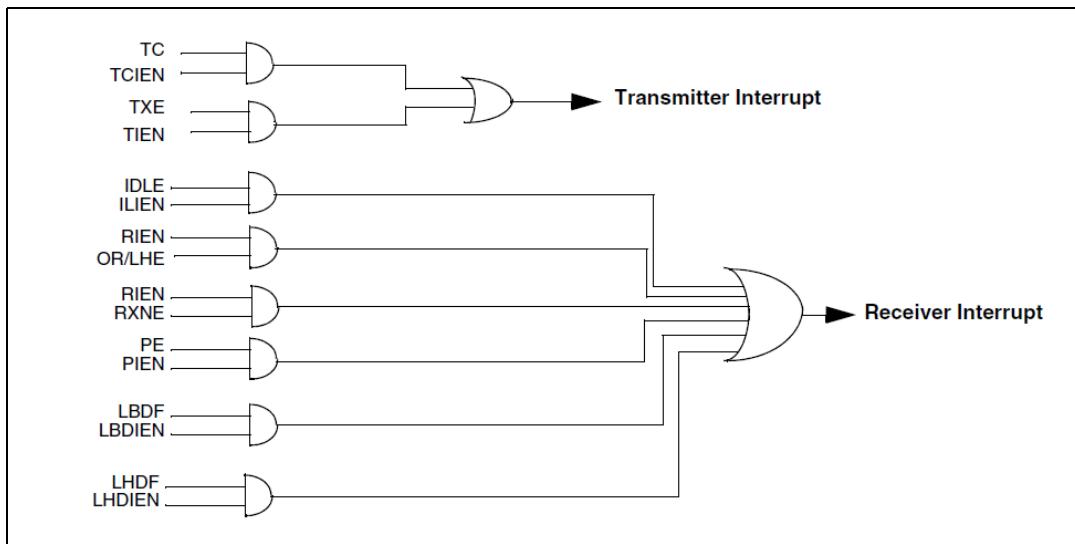
## 22.6 UART interrupts

**Table 60. UART 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 / LIN header error	OR/LHE		Yes	No
Idle line detected	IDLE	ILIEN	Yes	No
Parity error	PE	PIEN	Yes	No
Break flag	LBDF	LBDIEN	Yes	No
Header Flag	LHDF	LHDIEN	Yes	No

- Note:
- 1 *The UART interrupt events are connected to two interrupt vectors (see [Figure 139](#)).*
    - 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 CC register is reset (RIM instruction).*

Figure 139. UART interrupt mapping diagram



## 22.7 UART registers

### 22.7.1 Status register (UART\_SR)

Address offset: 0x00

Reset value: 0xC0

7	6	5	4	3	2	1	0
TXE	TC	RXNE	IDLE	OR/LHE	NF	FE	PE
r	rc_w0	rc_w0	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 UART\_CR2 register. It is cleared by a write to the UART\_DR register.

- 0: Data is not transferred to the shift register
- 1: Data is transferred to the shift register

Bit 6 **TC**: Transmission complete

This 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 UART\_CR2 register. The TC bit is cleared either by a software sequence (a read to the UART\_SR register followed by a write to the UART\_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 UART\_DR register. An interrupt is generated if RIEN=1 in the UART\_CR2 register. It is cleared by a read to the UART\_DR register. In UART2 and UART3, it can also be cleared by writing 0.

- 0: Data is not received
- 1: Received data is ready to be read.

Bit 4 **IDLE**: IDLE line detected <sup>(1)</sup>

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the ILIEN=1 in the UART\_CR2 register. It is cleared by a software sequence (a read to the UART\_SR register followed by a read to the UART\_DR register).

- 0: No Idle Line is detected
- 1: Idle Line is detected

Bit 3 **OR**: Overrun error<sup>(2)</sup>

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 UART\_CR2 register. It is cleared by a software sequence (a read to the UART\_SR register followed by a read to the UART\_DR register).

- 0: No Overrun error
- 1: Overrun error is detected

**LHE** LIN Header Error (LIN slave mode)

During LIN Header reception, this bit signals three error types:

- Break delimiter too short
- Synch Field error
- Deviation error (if LASE=1)
- Identifier framing error

- 0: No LIN Header error
- 1: LIN Header error detected

Bit 2 **NF**: Noise flag<sup>(3)</sup>

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 UART\_SR register followed by a read to the UART\_DR register).

- 0: No noise is detected
- 1: Noise is detected

Bit 1 **FE**: Framing error<sup>(4)</sup>

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 UART\_SR register followed by a read to the UART\_DR register).

- 0: No Framing error is detected
- 1: Framing error or break character is detected

*Note: For the UART2 and UART3, in LIN slave mode (bits LINE and LSLV are set), when a framing error is detected in the Synch or Identifier Fields , the FE bit is set. But the FE bit will not be set when a Break reception 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 UART\_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 UART\_CR1 register.

- 0: No parity error
- 1: Parity error (or, in LIN slave mode, identifier parity error)

1. The IDLE bit will not be 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 will not be lost but the shift register will be overwritten.
3. This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt.
4. This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. If the word currently being transferred causes both frame error and overrun error, it will be transferred and only the OR bit will be set.

## 22.7.2 Data register (UART\_DR)

Address offset: 0x01

Reset value: 0XX

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

## 22.7.3 Baud rate register 1 (UART\_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 119: How to code UART\\_DIV in the BRR registers on page 336](#) and [Table 54: Baud rate programming and error calculation on page 337](#)

*Note:* 1 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
UART_DIV[11:4]							
rw	rw	rw	rw	-	rw	rw	rw

Bits 7:0 **UART\_DIV[11:4]** UART\_DIV bits <sup>(1)</sup>

These 8 bits define the 2nd and 3rd nibbles of the 16-bit UART divider (UART\_DIV).

1. BRR1 = 00h means UART clock is disabled.

## 22.7.4 Baud rate register 2 (UART\_BRR2)

Address offset: 0x03

Reset value: 0x00

7	6	5	4	3	2	1	0
UART_DIV[15:12]				UART_DIV[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:4 **UART\_DIV[15:12]** MSB of UART\_DIV.

These 4 bits define the MSB of the UART Divider (UART\_DIV)

Bits 3:0 **UART\_DIV[3:0]**: LSB of UART\_DIV.

These 4 bits define the LSB of the UART Divider (UART\_DIV)

## 22.7.5 Control register 1 (UART\_CR1)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
R8	T8	UARTD	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 **UARTD**: UART Disable (for low power consumption).

When this bit is set the UART 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: UART enabled

1: UART 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 UART\_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) In LIN slave mode, the M bit and the STOP[1:0] bits in the UART\_CR3 register should be kept at 0.*

Bit 3 **WAKE**: Wakeup method.

This bit determines the UART wakeup method, it is set or cleared by software.

0: Idle Line

1: Address Mark

Bit 2 **PCEN**: Parity control enable.

– UART Mode

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

– LIN slave mode

This bit enables the LIN identifier parity check while the UART is in LIN slave mode.

0: Identifier parity check disabled

1: Identifier parity check enabled

Bit 1 **PS**: Parity selection.

This bit selects the odd or even parity when the parity generation/detection is enabled (PCEN bit set) in UART mode. 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 UART\_SR register

## 22.7.6 Control register 2 (UART\_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 UART interrupt is generated whenever TXE=1 in the UART\_SR register

Bit 6 **TCIEN**: Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever TC=1 in the UART\_SR register

Bit 5 **RIEN**: Receiver interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever OR=1 or RXNE=1 in the UART\_SR register

Bit 4 **ILIEN**: IDLE Line interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever IDLE=1 in the UART\_SR register

**Bit 3 TEN:** Transmitter enable <sup>(1) (2)</sup>

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

## – UART mode

This bit determines if the UART 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.<sup>(3) (4)</sup>

## – LIN slave mode (UART2, UART3 and UART4 only, if bits LINE and LSLV are set)

While LIN is used in slave mode, setting the RWU bit allows the detection of Headers only and prevents the reception of any other characters. Refer to [Mute mode and errors on page 355](#). In LIN slave mode, when RXNE is set, the software can not set or clear the RWU bit.

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 UART 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.

### 22.7.7 Control register 3 (UART\_CR3)

Address offset: 0x06

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL
r	rw	rw	rw	rw	rw	rw	rw

Bit 7 Reserved, must be kept cleared.

Bit 6 **LINEN**: LIN mode enable

This bit is set and cleared by software.

0: LIN mode disabled

1: LIN mode enabled

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: 1.5 Stop bits

Note: For LIN slave mode, both bits should be kept cleared.

Bit 3 **CLKEN**: Clock enable

This bit allows the user to enable the SCLK pin.

0: SLK pin disabled

1: SLK pin enabled

*Note: This bit is not available for UART3.*

Bit 2 **CPOL**: Clock polarity <sup>(1)</sup>

This bit allows the user to select the polarity of the clock output on the SCLK pin. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: SCK to 0 when idle

1: SCK to 1 when idle.

*Note: This bit is not available for UART3.*

Bit 1 **CPHA**: Clock phase <sup>(1)</sup>

This bit allows the user to select the phase of the clock output on the SCLK 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

*Note: This bit is not available for UART3.*

Bit 0 **LBCL**: Last bit clock pulse.<sup>(1)(2)</sup>

This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the SCLK pin.

0: The clock pulse of the last data bit is not output to the SCLK pin.

1: The clock pulse of the last data bit is output to the SCLK pin.

*Note: This bit is not available for UART3.*

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 UART\_CR1 register.

## 22.7.8 Control register 4 (UART\_CR4)

Address offset: 0x07

Reset value: 0x00

	7	6	5	4	3	2	1	0
	Reserved	LBDIEN	LBDL	LBDF	ADD[3:0]			
	r	rw	rw	rw	rw	rw	rw	rw

Bit 7 Reserved, must be kept cleared.

Bit 6 **LBDIEN**: LIN Break Detection Interrupt Enable.

Break interrupt mask (break detection using break delimiter).

0: LIN break detection interrupt disabled

1: LIN break detection interrupt enabled

Bit 5 **LBDL**: LIN Break Detection Length.

This bit is for selection between 11 bit or 10 bit break detection.

0: 10 bit break detection

1: 11 bit break detection

Bit 4 **LBDF**: LIN Break Detection Flag.

LIN Break Detection Flag (Status flag)

This bit is set by hardware and cleared by software writing 0.

0: LIN Break not detected

1: LIN Break detected

An interrupt is generated when LBDF=1 if LBDIEN=1

Bits 3:0 **ADD[3:0]**: Address of the UART node.

This bitfield gives the address of the UART node.

This is used in multi-processor communication during mute mode, for wakeup with address mark detection.

### 22.7.9 Control register 5 (UART\_CR5)

Address offset: 0x08

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	SCEN	NACK	HDSEL	IRLP	IREN	Reserved	
r	r	r	rw	rw	rw		r

Bits 7:6 Reserved, must be kept cleared.

Bit 5 **SCEN**: Smartcard mode enable.

This bit is used for enabling Smartcard mode.

0: Smartcard Mode disabled

1: Smartcard Mode enabled

*Note: This bit is not available for UART3.*

Bit 4 **NACK**: Smartcard NACK enable

0: NACK transmission in case of parity error is disabled

1: NACK transmission during parity error is enabled.

*Note: This bit is not available for UART3.*

Bit 3 **HDSEL**: Half-Duplex Selection

Selection of Single-wire Half-duplex mode

0: Half duplex mode is not selected

1: Half duplex mode is selected

*Note: This bit is not available for UART2 and UART3.*

Bit 2 **IRLP**: IrDA Low Power

This bit is used for selected between normal and Low power IrDA mode

0: Normal mode

1: Low power mode

*Note: This bit is not available for UART3.*

Bit 1 **IREN**: IrDA mode Enable

This bit is set and cleared by software.

0: IrDA disabled

1: IrDA enabled

*Note: This bit is not available for UART3.*

Bit 0 Reserved, must be kept cleared.

### 22.7.10 Control register 6 (UART\_CR6)

Address offset: 0x09

Reset value: 0x00

7	6	5	4	3	2	1	0
LDUM	Reserved	LSLV	LASE	Reserved	LHDIEN	LHDF	LSF
rw	r	rw	rw	r	rw	rc_w0	rc_w0

**Note:** This register is not available for UART1.

Bit 7 **LDUM**: LIN Divider Update Method

0: LDIV is updated as soon as BRR1 is written (if no automatic resynchronization update occurs at the same time).

1: LDIV is updated at the next received character (when RXNE=1) after a write to the BRR1 register.  
LDIV is coded using the two register BRR1 and BRR2

This bit is reset by hardware once LDIV is updated with the measured baud rate at the end of the synch field.

Bit 6 Reserved

Bit 5 **LSLV**: LIN Slave Enable

0: LIN Master Mode  
1: LIN Slave Mode

Bit 4 **LASE**: LIN automatic resynchronisation enable

0: LIN automatic resynchronization disabled  
1: LIN automatic resynchronization enabled

Bit 3 Reserved

Bit 2 **LHDIEN**: LIN Header Detection Interrupt Enable.

Header interrupt mask.  
0: LIN header detection interrupt disabled  
1: LIN header detection interrupt enabled

Bit 1 **LHDF**: LIN Header Detection Flag.

This bit is set by hardware when a LIN header is detected in LIN slave mode and cleared by software writing 0.

0: LIN Header not detected  
1: LIN Header detected (Break+Sync+Ident)  
An interrupt is generated when LHDF=1 if LHDIEN=1

Bits 0 **LSF**: LIN Sync Field

This bit indicates that the LIN Synch Field is being analyzed. It is only used in LIN Slave mode. In automatic resynchronization mode (LASE bit=1), when the UART is in the LIN Synch Field State it waits or counts the falling edges on the RDI line.

It is set by hardware as soon as a LIN Break is detected and cleared by hardware when the LIN Synch Field analysis is finished. This bit can also be cleared by software writing 0 to exit LIN Synch State and return to idle mode.

0: The current character is not the LIN Synch Field  
1: LIN Synch Field State (LIN Synch Field undergoing analysis)

### 22.7.11 Guard time register (UART\_GTR)

Address offset: 0x09 (UART1), 0x0A (UART2 and UART4)

Reset value: 0x00

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

Bits 7:0 **GT[7:0]**: Guard time value.

This register gives the Guard time value in terms of number of baud clocks.

This is used in Smartcard mode. The Transmission Complete flag is set after this guard time value.

*Note:* These bits are not available for UART3.

## 22.7.12 Prescaler register (UART\_PSCR)

Address offset: 0x0A (UART1), 0x0B (UART2 and UART4)

Reset value: 0x00

**Note:** *Care must be taken to program this register with correct value, when both Smartcard and IrDA interfaces are used in the application*

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

Bits 7:0 **PSC[7:0]: Prescaler value.**

- In IrDA Low Power mode

PSC[7:0] = IrDA Low Power Baud Rate <sup>(1)</sup>

Used for programming the prescaler for dividing the system clock to achieve the low power frequency:

The source clock is divided by the value given in the register (8 significant bits):

0000 0000: Reserved - do not program this value

0000 0001: divides the source clock by 1

0000 0010: divides the source clock by 2

...

- In Smartcard mode

PSC[4:0]: Prescaler value. <sup>(2) (3)</sup>

Used for programming the prescaler for dividing the system clock to provide the smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

0 0000: Reserved - do not program this value

0 0001: divides the source clock by 2

0 0010: divides the source clock by 4

0 0011: divides the source clock by 6

...

**Note:** These bits are not available for UART3.

1. This prescaler setting has no effect if IrDA mode is not enabled.
2. This prescaler setting has no effect if Smartcard mode is not enabled.
3. Bits [7:5] have no effect even if Smartcard mode is enabled.

### 22.7.13 UART register map and reset values

Table 61. UART1 register map

Address	Register name	7	6	5	4	3	2	1	0	
0x00	UART1_SR Reset Value	TXE 1	TC 1	RXNE 0	IDLE 0	OR 0	NF 0	FE 0	PE 0	
0x01	UART1_DR Reset Value	DR7 X	DR6 X	DR5 X	DR4 X	DR3 X	DR2 X	DR1 X	DR0 X	
0x02	UART1_BRR1 Reset Value	UART_DIV[11:4] 00000000								
0x03	UART1_BRR2 Reset Value	UART_DIV[15:12] 0000					UART_DIV[3:0] 0000			
0x04	UART1_CR1 Reset Value	R8 0	T8 0	UARTD 0	M 0	WAKE 0	PCEN 0	PS 0	PIEN 0	
0x05	UART1_CR2 Reset Value	TIEN 0	TCIEN 0	RIEN 0	IILEN 0	TEN 0	REN 0	RWU 0	SBK 0	
0x06	UART1_CR3 Reset Value	- 0	LINEN 0	STOP 00		CKEN 0	CPOL 0	CPHA 0	LBCL 0	
0x07	UART1_CR4 Reset Value	- 0	LBDIEN 0	LBDL 0	LBDF 0	ADD[3:0] 0000				
0x08	UART1_CR5 Reset Value	- 0	- 0	SCEN 0	NACK 0	HDSEL 0	IRLP 0	IREN 0	0	
0x09	UART1_GTR Reset Value	GT7 0	GT6 0	GT5 0	GT4 0	GT3 0	GT2 0	GT1 0	GT0 0	
0x0A	UART1_PSCR Reset Value	PSC7 0	PSC6 0	PSC5 0	PSC4 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0	

Table 62. UART2 register map

Address	Register name	7	6	5	4	3	2	1	0	
0x00	UART2_SR Reset Value	TXE 1	TC 1	RXNE 0	IDLE 0	OR 0	NF 0	FE 0	PE 0	
0x01	UART2_DR Reset Value	DR7 X	DR6 X	DR5 X	DR4 X	DR3 X	DR2 X	DR1 X	DR0 X	
0x02	UART2_BRR1 Reset Value	UART_DIV[11:4] 00000000								
0x03	UART2_BRR2 Reset Value	UART_DIV[15:12] 0000					UART_DIV[3:0] 0000			
0x04	UART2_CR1 Reset Value	R8 0	T8 0	UARTD 0	M 0	WAKE 0	PCEN 0	PS 0	PIEN 0	
0x05	UART2_CR2 Reset Value	TIEN 0	TCIEN 0	RIEN 0	IILEN 0	TEN 0	REN 0	RWU 0	SBK 0	
0x06	UART2_CR3 Reset Value	- 0	LINEN 0	STOP 00		CKEN 0	CPOL 0	CPHA 0	LBCL 0	

**Table 62. UART2 register map (continued)**

<b>Address</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x07	UART2_CR4 Reset Value	-0	LBDIEN0	LBDL0	LBDF0		ADD[3:0] 0000		
0x08	UART2_CR5 Reset Value	-0	-0	SCEN0	NACK0	-0	IRLP0	IREN0	0
0x09	UART2_CR6 Reset Value	LDUM0	-0	LSLV0	LASE0	-0	LHDIEN0	LHDF0	LSF0
0x0A	UART2_GTR Reset Value	GT70	GT60	GT50	GT40	GT30	GT20	GT10	GT00
0x0B	UART2_PSCR Reset Value	PSC70	PSC60	PSC50	PSC40	PSC30	PSC20	PSC10	PSC00

**Table 63. UART3 register map**

<b>Address</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x00	UART3_SR Reset Value	TXE1	TC1	RXNE0	IDLE0	OR0	NF0	FE0	PE0
0x01	UART3_DR Reset Value	DR7X	DR6X	DR5X	DR4X	DR3X	DR2X	DR1X	DR0X
0x02	UART3_BRR1 Reset Value					UART_DIV[11:4] 00000000			
0x03	UART3_BRR2 Reset Value			UART_DIV[15:12] 0000			UART_DIV[3:0] 0000		
0x04	UART3_CR1 Reset Value	R80	T80	UARTD0	M0	WAKE0	PCEN0	PS0	PIEN0
0x05	UART3_CR2 Reset Value	TIEN0	TCIEN0	RIEN0	ILIEN0	TEN0	REN0	RWU0	SBK0
0x06	UART3_CR3 Reset Value	-0	LINEN0		STOP00	-0	-0	-0	-0
0x07	UART3_CR4 Reset Value	-0	LBDIEN0	LBDL0	LBDF0		ADD[3:0] 0000		
0x08	Reserved								
0x09	UART2_CR6 Reset Value	LDUM0	-0	LSLV0	LASE0	-0	LHDIEN0	LHDF0	LSF0

**Table 64. UART4 register map**

<b>Address</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x00	UART4_SR Reset Value	TXE 1	TC 1	RXNE 0	IDLE 0	OR 0	NF 0	FE 0	PE 0
0x01	UART4_DR Reset Value	DR7 X	DR6 X	DR5 X	DR4 X	DR3 X	DR2 X	DR1 X	DR0 X
0x02	UART4_BRR1 Reset Value					UART_DIV[11:4] 00000000			
0x03	UART4_BRR2 Reset Value			UART_DIV[15:12] 0000			UART_DIV[3:0] 0000		
0x04	UART4_CR1 Reset Value	R8 0	T8 0	UARTD 0	M 0	WAKE 0	PCEN 0	PS 0	PIEN 0
0x05	UART4_CR2 Reset Value	TIEN 0	TCIEN 0	RIEN 0	IILEN 0	TEN 0	REN 0	RWU 0	SBK 0
0x06	UART4_CR3 Reset Value	- 0	LINEN 0		STOP 00	CKEN 0	CPOL 0	CPHA 0	LBCL 0
0x07	UART4_CR4 Reset Value	- 0	LBDIEN 0	LBDL 0	LBDF 0			ADD[3:0] 0000	
0x08	UART4_CR5 Reset Value	- 0	- 0	SCEN 0	NACK 0	HDSEL 0	IRLP 0	IREN 0	0
0x09	UART4_CR6 Reset Value	LDUM 0	- 0	LSLV 0	LASE 0	- 0	LHDIEN 0	LHDF 0	LSF 0
0x0A	UART4_GTR Reset Value	GT7 0	GT6 0	GT5 0	GT4 0	GT3 0	GT2 0	GT1 0	GT0 0
0x0B	UART4_PSCR Reset Value	PSC7 0	PSC6 0	PSC5 0	PSC4 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0

## 23 Controller area network (beCAN)

### 23.1 Introduction

The Basic Enhanced CAN peripheral, named beCAN, interfaces the CAN network. It supports the CAN protocol version 2.0A and B. It has been designed to manage high number of incoming messages efficiently with a minimum CPU load. It also meets the priority requirements for transmit messages.

For safety-critical applications the CAN controller provides all hardware functions for supporting the CAN Time triggered Communication option.

### 23.2 beCAN main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

#### Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

#### Reception

- One receive FIFO with three stages
- Six scalable filter banks
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

#### Time triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Configurable timer resolution
- Time Stamp sent in last two data bytes

#### Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

### 23.3 beCAN general description

In today's CAN applications, the number of nodes in a network is increasing and often several networks are linked together via gateways. Typically the number of messages in the system (and thus to be handled by each node) has significantly increased. In addition to the application messages, Network Management and Diagnostic messages have been introduced.

- An enhanced filtering mechanism is required to handle each type of message.

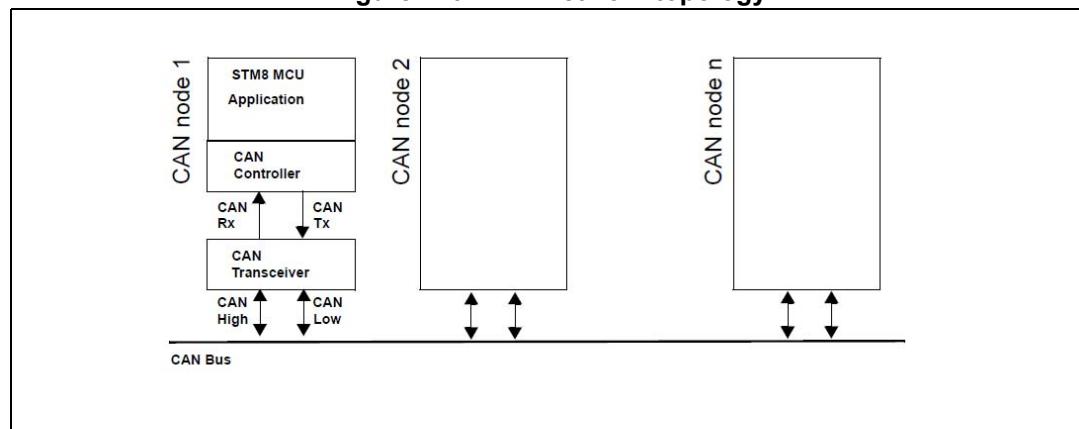
Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception have to be reduced.

- A receive FIFO scheme allows the CPU to be dedicated to application tasks for a long time period without losing messages.

The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.

- All mailboxes and registers are organized in 16-byte pages mapped at the same address and selected via a page select register.

**Figure 140. CAN network topology**



#### 23.3.1 CAN 2.0B active core

The beCAN module handles the transmission and the reception of CAN messages fully autonomously. Standard identifiers (11-bit) and extended identifiers (29-bit) are fully supported by hardware.

#### 23.3.2 Control, status and configuration registers

The application uses these registers to:

- Configure CAN parameters, e.g. baud rate
- Request transmissions
- Handle receptions
- Manage interrupts
- Get diagnostic information

### 23.3.3 Tx mailboxes

Three transmit mailboxes are provided to the software for setting up messages. The Transmission Scheduler decides which mailbox has to be transmitted first.

### 23.3.4 Acceptance filters

The beCAN provides six scalable/configurable identifier filter banks for selecting the incoming messages the software needs and discarding the others.

#### Receive FIFO

The receive FIFO is used by the CAN controller to store the incoming messages. Three complete messages can be stored in the FIFO. The software always accesses the next available message at the same address. The FIFO is managed completely by hardware.

**Figure 141. beCAN block diagram**

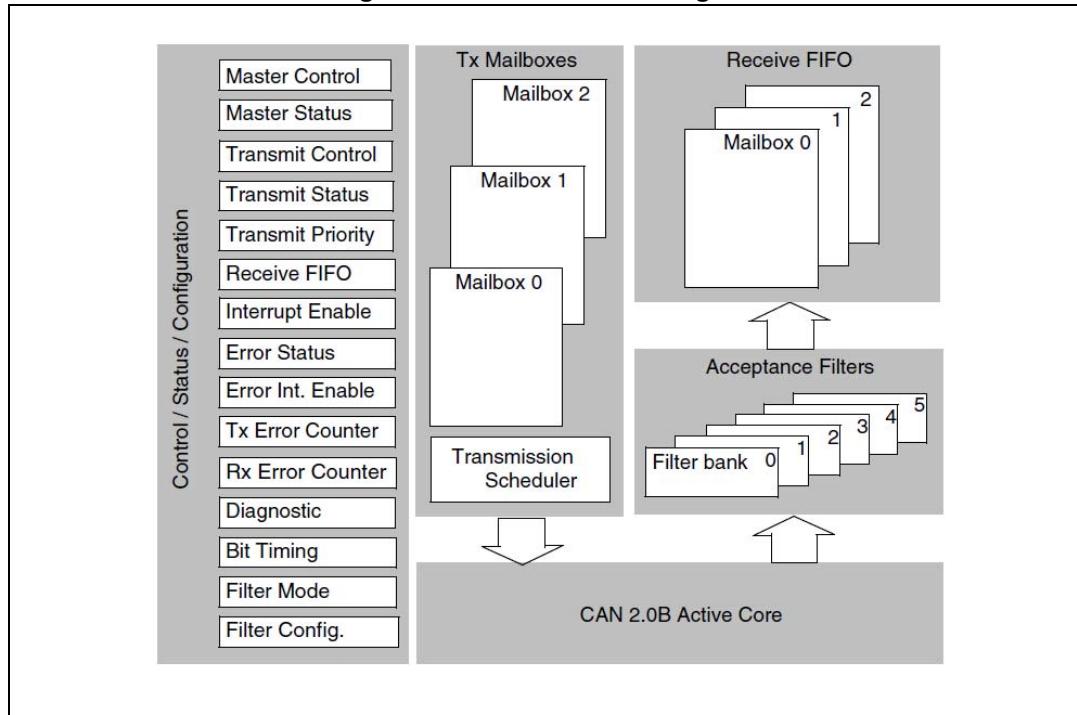
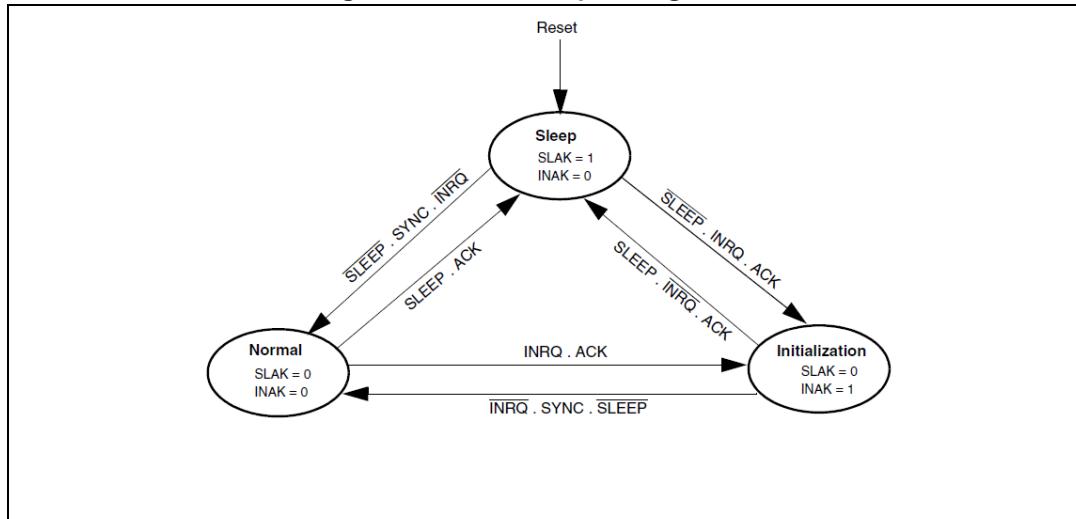


Figure 142. beCAN operating modes



## 23.4 Operating modes

beCAN has three main operating modes: **Initialization**, **Normal** and **Sleep**. After a hardware reset, beCAN is in sleep mode to reduce power consumption. The software requests beCAN to enter **Initialization** or **Sleep** mode by setting the INRQ or SLEEP bits in the CAN\_MCR register. Once the mode has been entered, beCAN confirms it by setting the INAK or SLAK bits in the CAN\_MSR register. When neither INAK nor SLAK are set, beCAN is in **Normal** mode. Before entering **Normal** mode beCAN always has to **synchronize** on the CAN bus. To synchronize, beCAN waits until the CAN bus is idle, this means 11 consecutive recessive bits have been monitored on CANRX.

**Note:** *The beCAN controls the PG0 port (CAN\_TX pin) when the beCAN is in Normal mode, whereas the PG0 port is controlled by the PG\_ODR and PG\_DDR GPIO registers while the beCAN is in Initialization mode or in Sleep mode. Therefore the PG\_ODR and PG\_DDR registers must be configured in order to maintain a recessive state on CAN\_TX pin.*

### 23.4.1 Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter this mode the software sets the INRQ bit in the CAN\_MCR register and waits until the hardware has confirmed the request by setting the INAK bit in the CAN\_MSR register.

To leave Initialization mode, the software clears the INRQ bit. beCAN has exit Initialization mode once the INAK bit has been cleared by hardware. However the Rx line has to be in recessive state to leave this mode.

While in Initialization mode, all message transfers to and from the CAN bus are stopped and the status of the CAN bus output CANTX is recessive (high).

Entering Initialization Mode does not change any of the configuration registers.

To initialize the CAN Controller, software has to set up the Bit Timing registers and the filter banks. If a filter bank is not used, it is recommended to leave it non active (leave the corresponding FACT bit in the CAN\_FCRx register cleared).

### 23.4.2 Normal mode

Once the initialization has been done, the software must request the hardware to enter Normal mode, to synchronize on the CAN bus and start reception and transmission. This request to enter Normal mode is done by clearing the INRQ bit in the CAN\_MCR register. Afterwards, the beCAN is synchronized with the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle state) before finishing the switch to Normal mode and being ready to take part in bus activities. The switch completion is confirmed by hardware by clearing the INAK bit in the CAN\_MSR register.

The initialization of the filter values is independent from Initialization mode but must be done while the filter bank is not active (corresponding FACTx bit cleared). The filter bank scale and mode configuration must be configured in initialization mode.

### 23.4.3 Sleep mode (low power)

To reduce power consumption, beCAN has a low power mode called Sleep mode. This mode is entered on software request by setting the SLEEP bit in the CAN\_MCR register. In this mode, the beCAN clock is stopped, however software can still access the beCAN mailboxes.

**Note:** *If software requests entry to initialization mode by setting the INRQ bit while beCAN is in sleep mode, it must also clear the SLEEP bit.*

beCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

On CAN bus activity detection, hardware automatically performs the wakeup sequence by clearing the SLEEP bit if the AWUM bit in the CAN\_MCR register is set. If the AWUM bit is cleared, software has to clear the SLEEP bit when a wakeup interrupt occurs, in order to exit from sleep mode.

**Note:** *If the wakeup interrupt is enabled (WKUIE bit set in CAN\_IER register) a wakeup interrupt will be generated on detection of CAN bus activity, even if the beCAN automatically performs the wakeup sequence.*

After the SLEEP bit has been cleared, Sleep mode is exited once beCAN has synchronized with the CAN bus, refer to [Figure 142: beCAN operating modes](#). However the Rx line has to be in recessive state to leave this mode. Sleep mode is exited once the SLAK bit has been cleared by hardware.

### 23.4.4 Time triggered communication mode

The TTCM (Time Triggered Communication Mode in CAN\_MCR) bit has to be set to enable the Time Triggered Communication mechanism.

In this mode, the internal counter of the CAN hardware is activated and used to generate the Time Stamp value stored in the CAN\_MTSRH and CAN\_MTSRL registers (for Rx and Tx mailboxes). The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.

The TGT bit (Transmit Global Time in CAN\_MDLCR) enables automatic transmission of the contents of both CAN\_MTSRH and CAN\_MTSRL in the two last data bytes of the message (refer to the TTCAN specification ISO 11898-4).

## 23.5 Test modes

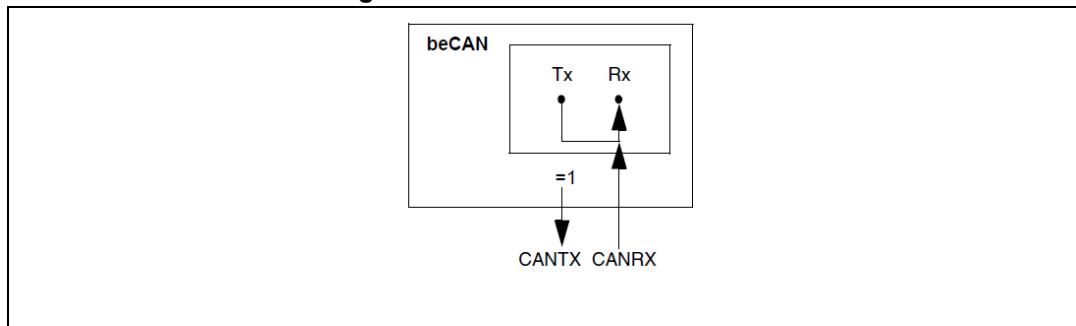
Test modes can be selected by the SLM and LBKM bits in the CAN\_DGR register. These bits must be configured while beCAN is in Initialization mode. Once a test mode has been selected, the INRQ bit in the CAN\_MCR register must be reset to enter Normal mode.

### 23.5.1 Silent mode

The beCAN can be put in Silent mode by setting the SLM bit in the CAN\_DGR register.

In Silent mode, the beCAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the beCAN has to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).

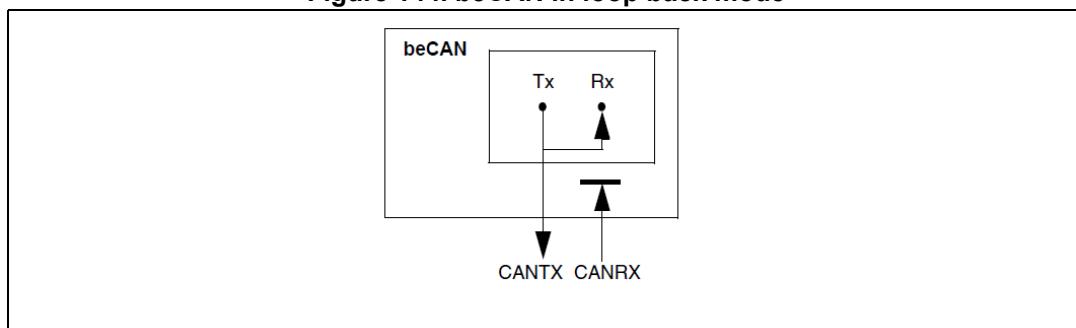
**Figure 143. beCAN in silent mode**



### 23.5.2 Loop back mode

The beCAN can be set in Loop Back Mode by setting the LBKM bit in the CAN\_DGR register. In Loop Back Mode, the beCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in the FIFO.

**Figure 144. beCAN in loop back mode**



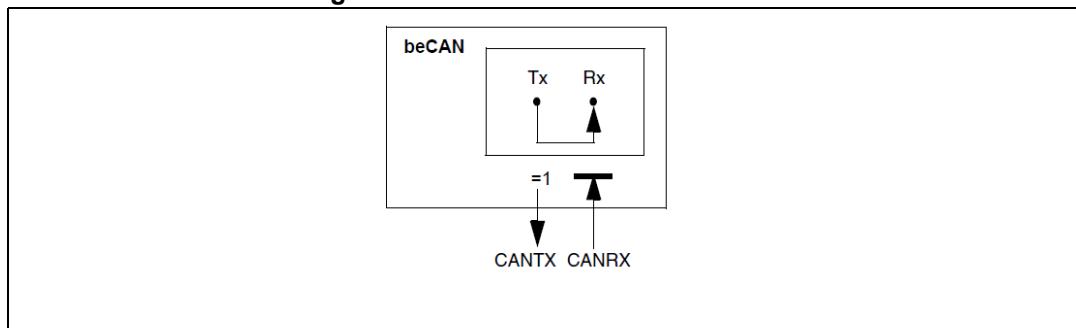
This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the beCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the beCAN. The transmitted messages can be monitored on the CANTX pin.

**Note:** As the Tx line is still active in this mode, be aware that it can disturb the communication on the CAN bus.

### 23.5.3 Loop back combined with silent mode

It is also possible to combine Loop Back mode and Silent mode by setting the LBKM and SILEM bits in the CAN\_DGR register. This mode can be used for a “Hot Selftest”, meaning the beCAN can be tested like in Loop Back mode but without affecting a running CAN system connected to the CANTX and CANRX pins. In this mode, the CANRX pin is disconnected from the beCAN and the CANTX pin is held recessive.

Figure 145. beCAN in combined mode



## 23.6 Functional description

### 23.6.1 Transmission handling

In order to transmit a message, the application must select one **empty** transmit mailbox, set up the identifier, the data length code (DLC) and the data before requesting the transmission by setting the corresponding TXRQ bit in the CAN\_MCSR register. Once the mailbox has exited **empty** state, the software no longer has write access to the mailbox registers.

Immediately after the TXRQ bit has been set, the mailbox enters **Pending** state and waits to become the highest priority mailbox, see *Transmit Priority*. As soon as the mailbox has the highest priority it will be **Scheduled** for transmission. The transmission of the message of the scheduled mailbox will start (enter **transmit** state) when the CAN bus becomes idle. Once the mailbox has been successfully transmitted, it will become **empty** again. The hardware indicates a successful transmission by setting the RQCP and TXOK bits in the CAN\_MCSR and CAN\_TSR registers.

If the transmission fails, the cause is indicated by the ALST bit in the CAN\_MCSR register in case of an Arbitration Lost, and/or the TERR bit, in case of transmission error detection.

#### Transmit priority

##### By identifier:

When more than one transmit mailbox is pending, the transmission order is given by the identifier of the message stored in the mailbox. The message with the lowest identifier value has the highest priority according to the arbitration of the CAN protocol. If the identifier values are equal, the lower mailbox number will be scheduled first.

**By transmit request order:**

The transmit mailboxes can be configured as a transmit FIFO by setting the TXFP bit in the CAN\_MCR register. In this mode the priority order is given by the transmit request order. This mode is very useful for segmented transmission.

**Abort**

A transmission request can be aborted by the user setting the ABRQ bit in the CAN\_MCSR register. In **pending** or **scheduled** state, the mailbox is aborted immediately. An abort request while the mailbox is in **transmit** state can have two results. If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN\_MCSR and CAN\_TSR registers. If the transmission fails, the mailbox becomes **scheduled**, the transmission is aborted and becomes **empty** with TXOK cleared. In all cases the mailbox will become **empty** again at least at the end of the current transmission.

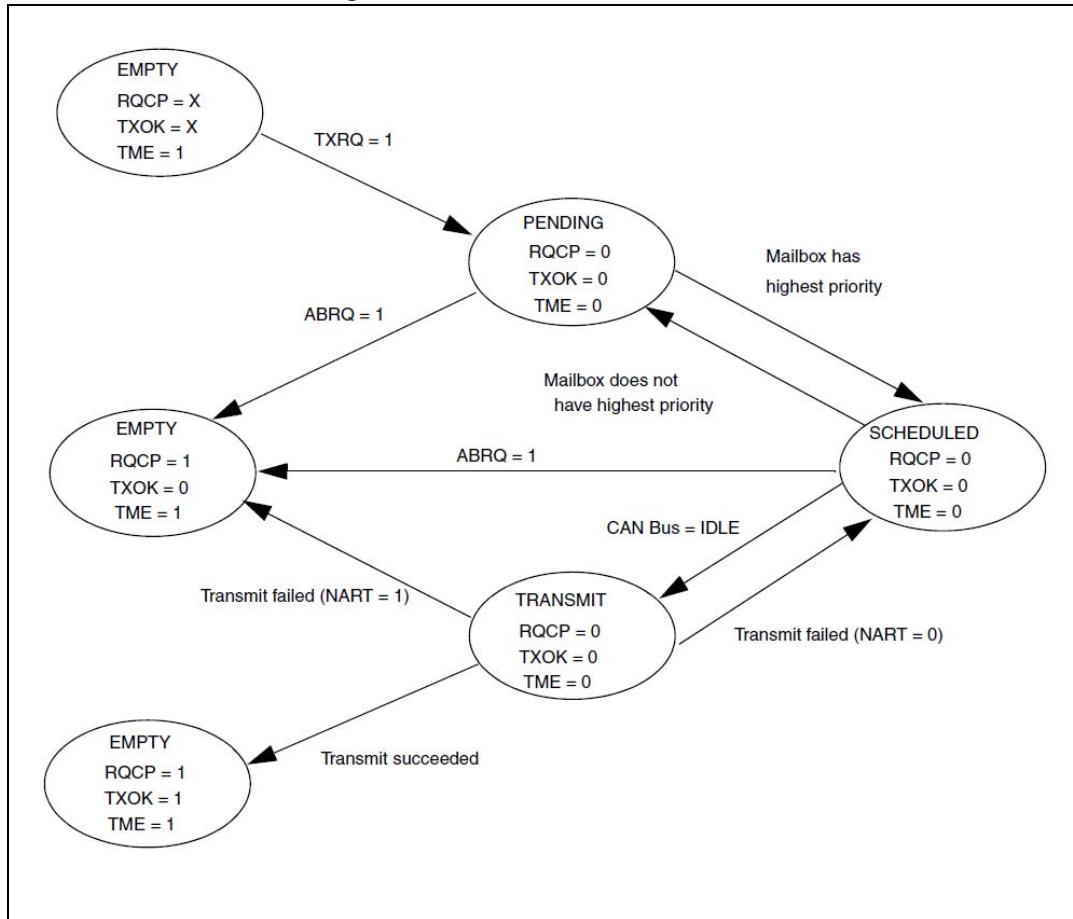
**Non-automatic retransmission mode**

This mode has been implemented in order to fulfil the requirement of the Time Triggered Communication option of the CAN standard. To configure the hardware in this mode the NART bit in the CAN\_MCR register must be set.

In this mode, each transmission is started only once. If the first attempt fails, due to an arbitration loss or an error, the hardware will not automatically restart the message transmission.

At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP bit in the CAN\_MCSR register. The result of the transmission is indicated in the CAN\_MCSR register by the TXOK, ALST and TERR bits.

Figure 146. Transmit mailbox states



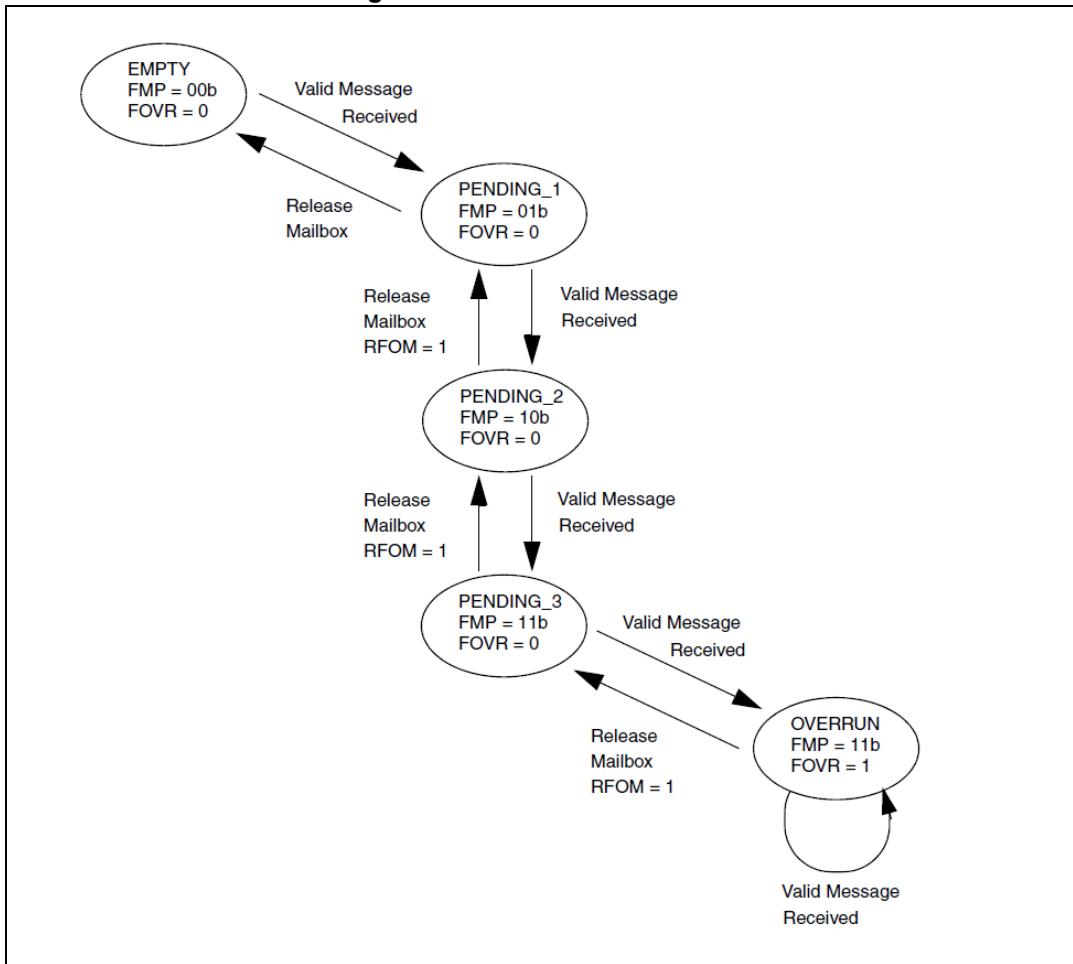
### 23.6.2 Reception handling

For the reception of CAN messages, three mailboxes organized as a FIFO are provided. In order to save CPU load, simplify the software and guarantee data consistency, the FIFO is managed completely by hardware. The application accesses the messages stored in the FIFO through the FIFO output mailbox.

#### Valid message

A received message is considered as valid **when** it has been received correctly according to the CAN protocol (no error until the last but one bit of the EOF field) **and** It passed through the identifier filtering successfully, see [Section 23.6.3: Identifier filtering](#).

**Figure 147. Receive FIFO states**



#### FIFO management

Starting from the **empty** state, the first valid message received is stored in the FIFO which becomes **pending\_1**. The hardware signals the event setting the FMP[1:0] bits in the CAN\_RFR register to the value 0b01. The message is available in the FIFO output mailbox. The software reads out the mailbox content and releases it by setting the RFOM bit in the CAN\_RFR register. The FIFO becomes **empty** again. If a new valid message has been received in the meantime, the FIFO stays in **pending\_1** state and the new message is available in the output mailbox.

If the application does not release the mailbox, the next valid message will be stored in the FIFO which enters **pending\_2** state ( $FMP[1:0] = 0b10$ ). The storage process is repeated for the next valid message putting the FIFO into **pending\_3** state ( $FMP[1:0] = 0b11$ ). At this point, the software must release the output mailbox by setting the RFOM bit, so that a mailbox is free to store the next valid message. Otherwise the next valid message received will cause a loss of message.

Refer also to [Section 23.6.4: Message storage](#).

### Overrun

Once the FIFO is in **pending\_3** state (i.e. the three mailboxes are full) the next valid message reception will lead to an **overrun** and a message will be lost. The hardware signals the overrun condition by setting the FOVR bit in the CAN\_RFR register. Which message is lost depends on the configuration of the FIFO:

- If the FIFO lock function is disabled (RFLM bit in the CAN\_MCR register cleared) the last message stored in the FIFO will be overwritten by the new incoming message. As a result, the last message is always available to the application.

**Note:**

*The previously received messages will stay in their positions in the FIFO, only the last one will be overwritten.*

- If the FIFO lock function is enabled (RFLM bit in the CAN\_MCR register set) the most recent message will be discarded and the software will have the three oldest messages in the FIFO available.

### Reception related interrupts

On the storage of the first message in the FIFO -  $FMP[1:0]$  bits change from  $0b00$  to  $0b01$  - an interrupt is generated if the FMPIE bit in the CAN\_IER register is set.

When the FIFO becomes full (i.e. a third message is stored) the FULL bit in the CAN\_RFR register is set and an interrupt is generated if the FFIE bit in the CAN\_IER register is set.

On overrun condition, the FOVR bit is set and an interrupt is generated if the FOVIE bit in the CAN\_IER register is set.

## 23.6.3 Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the RAM. If not, the message must be discarded without intervention by the software.

To fulfil this requirement, the beCAN Controller provides 6 configurable and scalable filter banks (5:0) in order to receive only the messages the software needs. This hardware filtering saves CPU resources which would be otherwise needed to perform filtering by software. Each filter bank x consists of eight 8-bit registers, CAN\_FxR[8:1].

### Scalable width

To optimize and adapt the filters to the application needs, each filter bank can be scaled independently. Depending on the filter scale a filter bank provides:

- One 32-bit filter for the STDID[10:0] / EXID[28:18], IDE, EXID[17:0] and RTR bits.
- Two 16-bit filters for the STDID[10:0] / EXID[28:18], RTR and IDE bits.
- Four 8-bit filters for the STDID[10:3] / EXID[28:21] bits. The other bits are considered as don't care.
- One 16-bit filter and two 8-bit filters for filtering the same set of bits as the 16 and 8-bit filters described above.

Refer to [Figure 148](#) through [Figure 151](#).

Furthermore, the filters can be configured in mask mode or in identifier list mode.

#### Mask mode

In **mask** mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as "must match" or as "don't care".

#### Identifier list mode

In **identifier list** mode, the mask registers are used as identifier registers. Thus instead of defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.

#### Filter bank scale and mode configuration

The filter banks are configured by means of the corresponding CAN\_FCRx register. To configure a filter bank this must be deactivated by clearing the FACT bit in the CAN\_FCRx register. The filter scale is configured by means of the FSC[1:0] bits in the corresponding CAN\_FCRx register. The **identifier list** or **identifier mask** mode for the corresponding Mask/Identifier registers is configured by means of the FMLx and FMHx bits in the CAN\_FMRx register. The FMLx bit defines the mode for the lower half (registers CAN\_FxR1-4), and the FMHx bit the mode for the upper half (registers CAN\_FxR5-8) of filter bank x. Refer to [Figure 148](#) through [Figure 151](#) for details.

#### Examples:

- If filter bank 1 is configured as two 16-bit filters, then the FML1 bit defines the mode of the CAN\_F1R3 and CAN\_F1R4 registers and the FMH1 bit defines the mode of the CAN\_F1R7 and CAN\_F1R8 registers.
- If filter bank 1 is configured as four 8-bit filters, then the FML1 bit defines the mode of the CAN\_F1R2 and CAN\_F1R4 registers and the FMH1 bit defines the mode of the CAN\_F1R6 and CAN\_F1R8 registers.

- Note:** In 32-bit configuration, the FMLx and FMHx bits must have the same value to ensure that the four Mask/Identifier registers are in the same mode.
- When a standard identifier is received (IDE bit is zero), the extended part of 32-bit or 16-bit filters is not compared.
- To filter a group of identifiers, configure the Mask/Identifier registers in mask mode.
- To select single identifiers, configure the Mask/Identifier registers in identifier list mode.
- Filters not used by the application should be left deactivated.
- Each filter within a filter bank is numbered (called the Filter Number) from 0 to a maximum dependent on the mode and the scale of each of the 6 filter banks.
- For the filter configuration, refer to [Figure 148](#) through [Figure 151](#).

**Figure 148. 32-bit filter bank configuration (FSCx bits = 0b11 in CAN\_FCRx register)**

Filter registers								Filter mode <sup>1</sup>								
Mapping	STID[10:3] / EXID[28:21]	STID [2:0] / EXID[20:18]	RTR	IDE	EXID [17:15]	EXID [14:7]	EXID[6:0]	0	FMHx = 0 FMLx = 0	FMHx = 1 FMLx = 1						
Identifier	CAN_FxR1	CAN_FxR2			CAN_FxR3	CAN_FxR4			ID	n						
Identifier/Mask	CAN_FxR5	CAN_FxR6			CAN_FxR7	CAN_FxR8			M	n+1						
ID= Identifier                    n = Filter number M = Mask                        x = Filter bank number																
1 The FMHx and FMLx bits are located in the CAN_FMR1 and CAN_FMR2 registers																

**Figure 149. 16-bit filter bank configuration (FSCx bits = 0b10 in CAN\_FCRx register)**

Filter registers								Filter mode <sup>1</sup>						
Mapping	STID[10:3] / EXID[28:21]	STID [2:0] / EXID [20:18]	RTR	IDE	EXID [17:15]	FMHx = 0 FMLx = 0	FMHx = 0 FMLx = 1	FMHx = 1 FMLx = 0	FMHx = 1 FMLx = 1					
Identifier	CAN_FxR1	CAN_FxR2			ID	n	ID	n	ID	n				
Identifier/Mask	CAN_FxR3	CAN_FxR4			M	n+1	ID	n+1	M	n+1	ID			
Identifier	CAN_FxR5	CAN_FxR6			ID	n+1	ID	n+1	ID	n+2	ID			
Identifier/Mask	CAN_FxR7	CAN_FxR8			M	n+2	ID	n+2	ID	n+2	ID			
ID= Identifier                    n = Filter number M = Mask                        x = Filter bank number														
1 The FMHx and FMLx bits are located in the CAN_FMR1 and CAN_FMR2 registers														

**Figure 150. 16/8-bit filter bank configuration (FSCx bits = 0b01 in CAN\_FCRx register)**

		Filter registers					Filter mode <sup>1</sup>					
Mapping		STID[10:3] / EXID[28:21]	STID [2:0] / EXID [20:18]	RTR	IDE	EXID [17:15]	FMHx = 0	FMLx = 0	FMHx = 1	FMLx = 0	FMHx = 1	FMLx = 1
Identifier	CAN_FxR1	CAN_FxR2					ID n	ID n	ID n	ID n		
Identifier/Mask	CAN_FxR3	CAN_FxR4		M			ID n+1	ID n+1	M	ID n+1	ID n+1	
Identifier	CAN_FxR5						ID n+1	ID n+2	ID n+1	ID n+2		
Identifier/Mask	CAN_FxR6			M			ID n+2	ID n+2	ID n+2	ID n+3		
Identifier	CAN_FxR7						ID n+2	ID n+3	ID n+3	ID n+4		
Identifier/Mask	CAN_FxR8			M			ID n+3	ID n+4	ID n+4	ID n+5		

ID= Identifier  
M = Mask  
n = Filter number  
x = Filter bank number

<sup>1</sup> The FMHx and FMLx bits are located in the CAN\_FMR1 and CAN\_FMR2 registers

**Figure 151. 8-bit filter bank configuration (FSCx bits = 0b00 in CAN\_FCRx register)**

		Filter registers					Filter mode <sup>1</sup>				
Mapping		STID[10:3] / EXID[28:21]	FMHx = 0 FMLx = 0	FMHx = 0 FMLx = 1	FMHx = 1 FMLx = 0	FMHx = 1 FMLx = 1	FMHx = 0	FMHx = 1	FMHx = 0	FMHx = 1	
Identifier	CAN_FxR1		ID n	ID n	ID n	ID n					
Identifier/Mask	CAN_FxR2		M	ID n+1	ID n+1	ID n+1	M	ID n+1	ID n+1	ID n+1	
Identifier	CAN_FxR3			ID n+2	ID n+2	ID n+2		ID n+2	ID n+2	ID n+2	
Identifier/Mask	CAN_FxR4		M	ID n+3	ID n+3	ID n+3	M	ID n+3	ID n+3	ID n+3	
Identifier	CAN_FxR5				ID n+4	ID n+4	ID n+4	ID n+4	ID n+4	ID n+4	
Identifier/Mask	CAN_FxR6		M		M	ID n+5	ID n+5	ID n+5	ID n+5	ID n+5	
Identifier	CAN_FxR7					ID n+5	ID n+6	ID n+6	ID n+6	ID n+6	
Identifier/Mask	CAN_FxR8		M			ID n+5	ID n+7	ID n+7	ID n+7	ID n+7	

ID= Identifier  
M = Mask  
n = Filter number  
x = Filter bank number

<sup>1</sup> The FMHx and FMLx bits are located in the CAN\_FMR1 and CAN\_FMR2 registers

### Filter match index

Once a message has been received in the FIFO it is available to the application. Typically application data are copied into RAM locations. To copy the data to the right location the application has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the CAN controller provides a Filter Match Index.

This index is stored in the mailbox together with the message according to the filter priority rules. Thus each received message has its associated Filter Match Index.

The Filter Match Index can be used in two ways:

- Compare the Filter Match Index with a list of expected values.
- Use the Filter Match Index as an index on an array to access the data destination location.

For non-masked filters, the software no longer has to compare the identifier.

If the filter is masked the software reduces the comparison to the masked bits only.

**Note:** *The index value of the filter number does not take into account the activation state of the filter banks.*

**Table 65. Example of filter numbering**

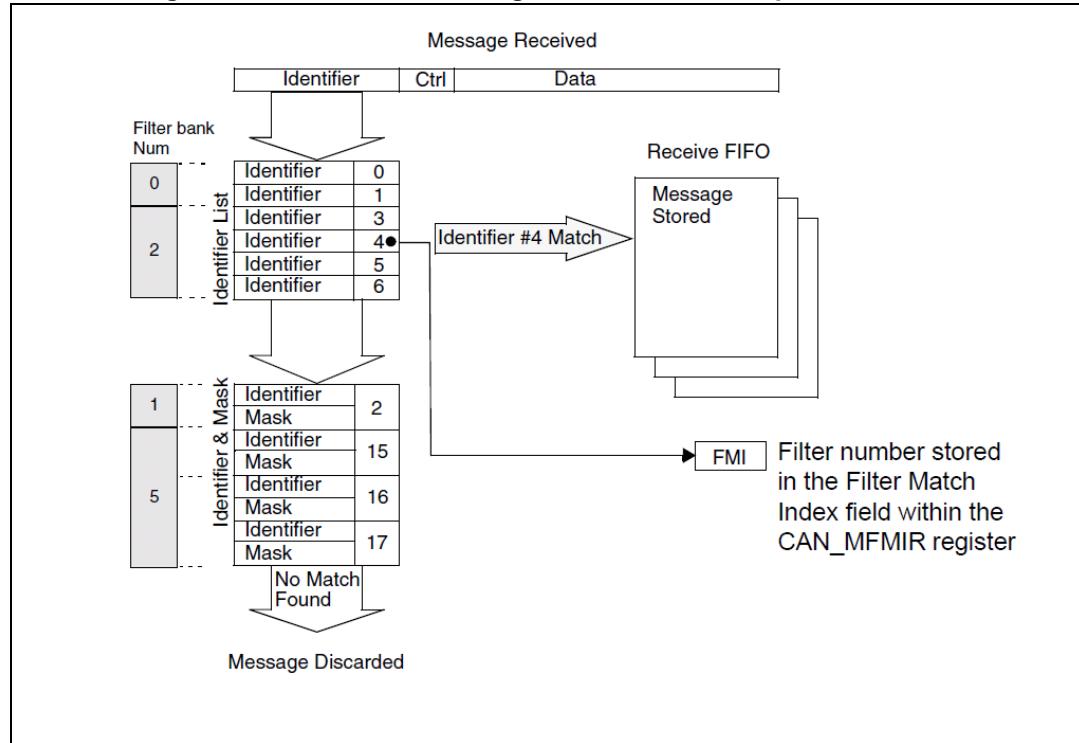
Filter bank						Filter number
Number	FCS	FMH	FML	FACT	Configuration	
0	0b11	1	1	1	Identifier list (32-bit)	0 1
1	0b11	0	0	1	Identifier mask (32-bit)	2
2	0b10	1	1	1	Identifier list (16-bit)	3 4 5 6
3	0b00	0	1	0	Deactivated Identifier List/Identifier mask (8-bit)	7 8 9 10 11 12
4	0b10	0	0	0	Deactivated Identifier Mask (16-bit)	13 14
5	0b01	0	0	1	Identifier Mask (16/8-bit)	15 16 17

### Filter priority rules

Depending on the filter combination it may occur that an identifier passes successfully through several filters. In this case the filter match value stored in the receive mailbox is chosen according to the following rules:

- A 32-bit filter takes priority over 16-bit filter which takes itself priority over 8-bit filter.
- For filters of equal scale, priority is given to the identifier List mode over the identifier Mask mode.
- For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority).

**Figure 152. Filter banks configured as in the example in [Table 65](#)**



The example above shows the filtering principle of the beCAN. On reception of a message, the identifier is compared first with the filters configured in identifier list mode. If there is a match, the message is stored in the FIFO and the index of the matching filter is stored in the Filter Match Index. As shown in the example, the identifier matches with Identifier #4 thus the message content and FMI 4 is stored in the FIFO.

If there is no match, the incoming identifier is then compared with the filters configured in mask mode.

If the identifier does not match any of the identifiers configured in the filters, the message is discarded by hardware without disturbing the software.

### 23.6.4 Message storage

The interface between the software and the hardware for the CAN messages is implemented by means of mailboxes. A mailbox contains all information related to a message; identifier, data, control, status and time stamp information.

#### Transmit mailbox

The software sets up the message to be transmitted in an empty transmit mailbox. The status of the transmission is indicated by hardware in the CAN\_MCSR register.

**Table 66. Transmit mailbox mapping**

Offset to Transmit Mailbox base address (bytes)	Register name
0	CAN_MCSR
1	CAN_MDLCR
2	CAN_MIDR1
3	CAN_MIDR2
4	CAN_MIDR3
5	CAN_MIDR4
6	CAN_MDAR1
7	CAN_MDAR2
8	CAN_MDAR3
9	CAN_MDAR4
10	CAN_MDAR5
11	CAN_MDAR6
12	CAN_MDAR7
13	CAN_MDAR8
14	CAN_MTSRL
15	CAN_MTSRH

### Receive mailbox

When a message has been received, it is available to the software in the FIFO output mailbox. Once the software has handled the message (e.g. read it) the software must release the FIFO output mailbox by means of the RFOM bit in the CAN\_RFR register to make the next incoming message available. The filter match index is stored in the CAN\_MFMIR register. The 16-bit time stamp value is stored in the CAN\_MTSRH and CAN\_MTSRL registers.

**Table 67. Receive mailbox mapping**

Offset to Receive Mailbox base address (bytes)	Register name
0	CAN_MFMIR
1	CAN_MDLCR
2	CAN_MIDR1
3	CAN_MIDR2
4	CAN_MIDR3
5	CAN_MIDR4
6	CAN_MDAR1
7	CAN_MDAR2
8	CAN_MDAR3
9	CAN_MDAR4
10	CAN_MDAR5
11	CAN_MDAR6
12	CAN_MDAR7
13	CAN_MDAR8
14	CAN_MTSRL
15	CAN_MTSRH

### 23.6.5 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (CAN\_TECR register) and a Receive Error Counter (CAN\_REC register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, please refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CAN\_ESR register. By means of CAN\_EIER register and ERRIE bit in CAN\_IER register, the software can configure the interrupt generation on error detection in a very flexible way.

#### Bus-Off recovery

The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF bit in CAN\_ESR register. In Bus-Off state, the beCAN is no longer able to transmit and receive messages.

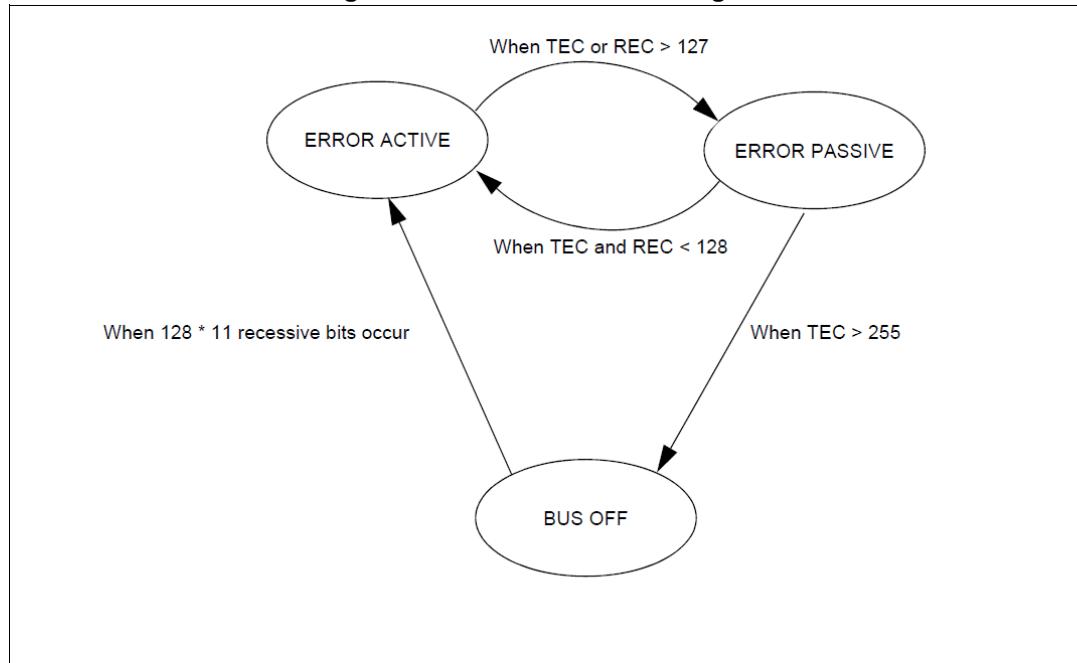
Depending on the ABOM bit in the CAN\_MCR register beCAN will recover from Bus-Off (become error active again) either automatically or on software request. But in both cases the beCAN has to wait at least for the recovery sequence specified in the CAN standard (128 x 11 consecutive recessive bits monitored on CANRX).

If ABOM is set, the beCAN will start the recovering sequence automatically after it has entered Bus-Off state.

If ABOM is cleared, the software must initiate the recovering sequence by requesting beCAN to enter initialization mode. Then beCAN starts monitoring the recovery sequence when the beCAN is requested to leave the initialization mode.

**Note:** *In initialization mode, beCAN does not monitor the CANRX signal, therefore it cannot complete the recovery sequence. To recover, beCAN must be in normal mode.*

Figure 153. CAN error state diagram



### 23.6.6 Bit timing

The bit timing logic monitors the serial bus-line and performs sampling and adjustment of the sample point by synchronizing on the start-bit edge and resynchronizing on the following edges. Its operation may be explained simply by splitting nominal bit time into three segments as follows:

- **Synchronization segment (SYNC\_SEG)**: a bit change is expected to occur within this time segment. It has a fixed length of one time quantum ( $1 \times t_{\text{q}}$ ).
- **Bit segment 1 (BS1)**: defines the location of the sample point. It includes the PROP\_SEG and PHASE\_SEG1 of the CAN standard. Its duration is programmable between 1 and 16 time quanta but may be automatically lengthened to compensate for positive phase drifts due to differences in the frequency of the various nodes of the network.
- **Bit segment 2 (BS2)**: defines the location of the transmit point. It represents the PHASE\_SEG2 of the CAN standard. Its duration is programmable between 1 and 8 time quanta but may also be automatically shortened to compensate for negative phase drifts.

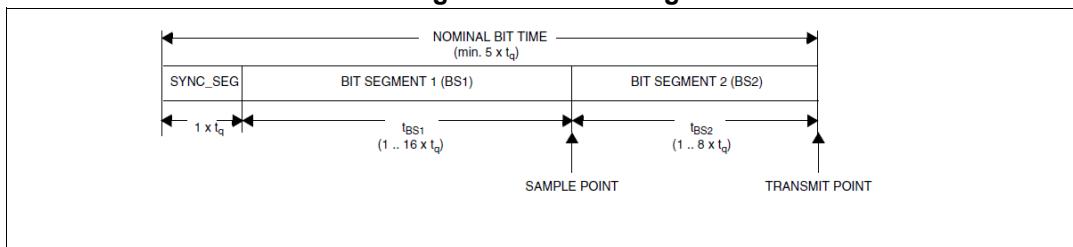
The resynchronization Jump Width (SJW) defines an upper bound to the amount of lengthening or shortening of the bit segments. It is programmable between 1 and 4 time quanta.

To guarantee the correct behavior of the CAN controller, SYNC\_SEG + BS1 + BS2 must be greater than or equal to 5 time quanta.

*Note:* For a detailed description of the CAN bit timing and resynchronization mechanism, please refer to the ISO 11898 standard.

As a safeguard against programming errors, the configuration of the Bit Timing Registers CAN\_BTR1 and CAN\_BTR2 is only possible while the device is in Initialization mode.

Figure 154. Bit timing



$$\text{BaudRate} = \frac{1}{\text{NominalBitTime}}$$

$$\text{NominalBitTime} = t_q + t_{BS1} + t_{BS2}$$

with

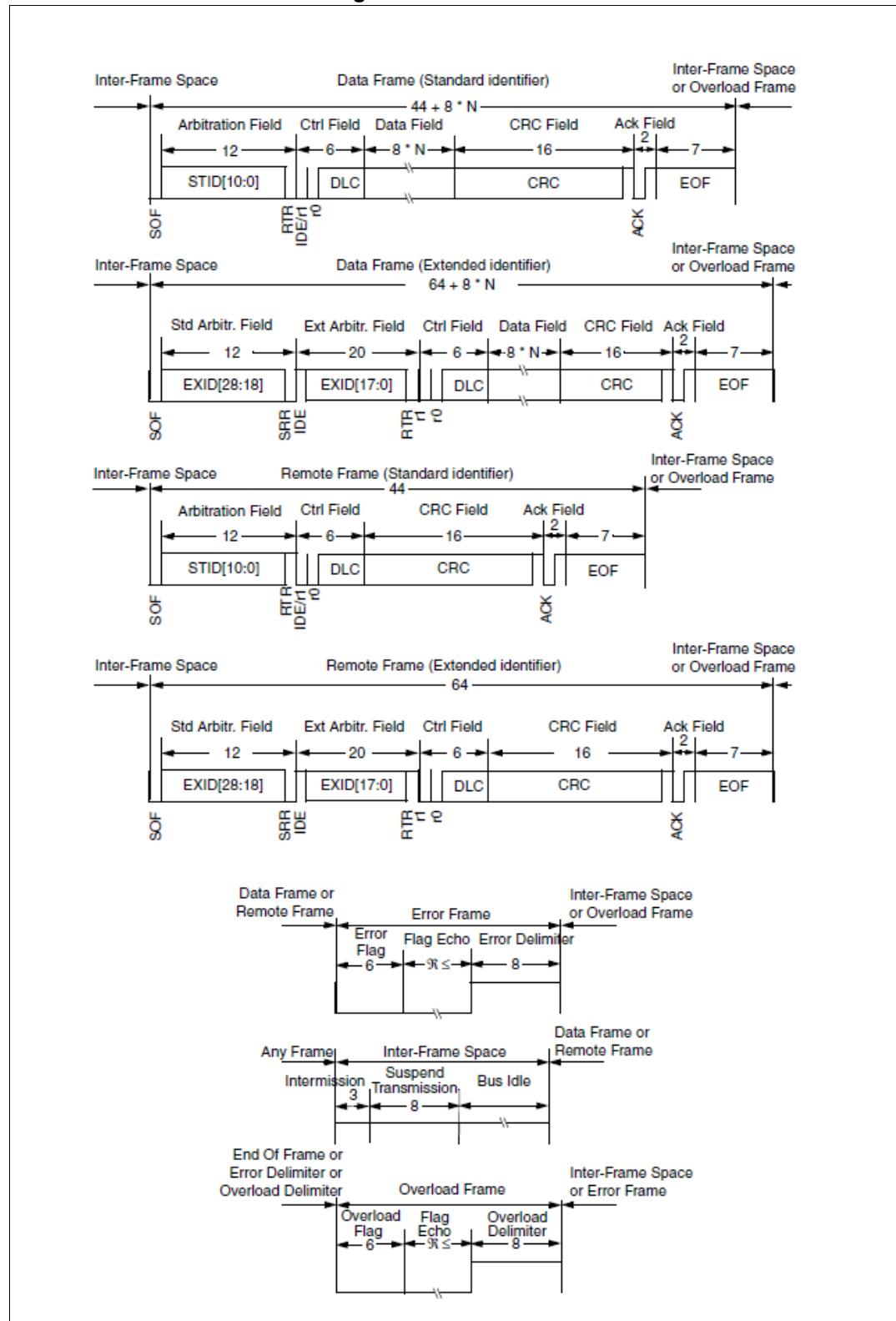
$$t_{BS1} = (\text{BS1}[3:0] + 1) \times t_q$$

$$t_{BS2} = (\text{BS2}[2:0] + 1) \times t_q$$

$$t_q = (\text{BRP}[5:0] + 1) \times t_{\text{sys}}$$

where  $t_q$  refers to the time quantum,  $t_{\text{sys}}$  is the system clock period ( $f_{\text{MASTER}}$ ). BRP[5:0], BS1[3:0], and BS2[2:0] are defined in the CAN\_BTR1 and CAN\_BTR2 registers.

Figure 155. CAN frames



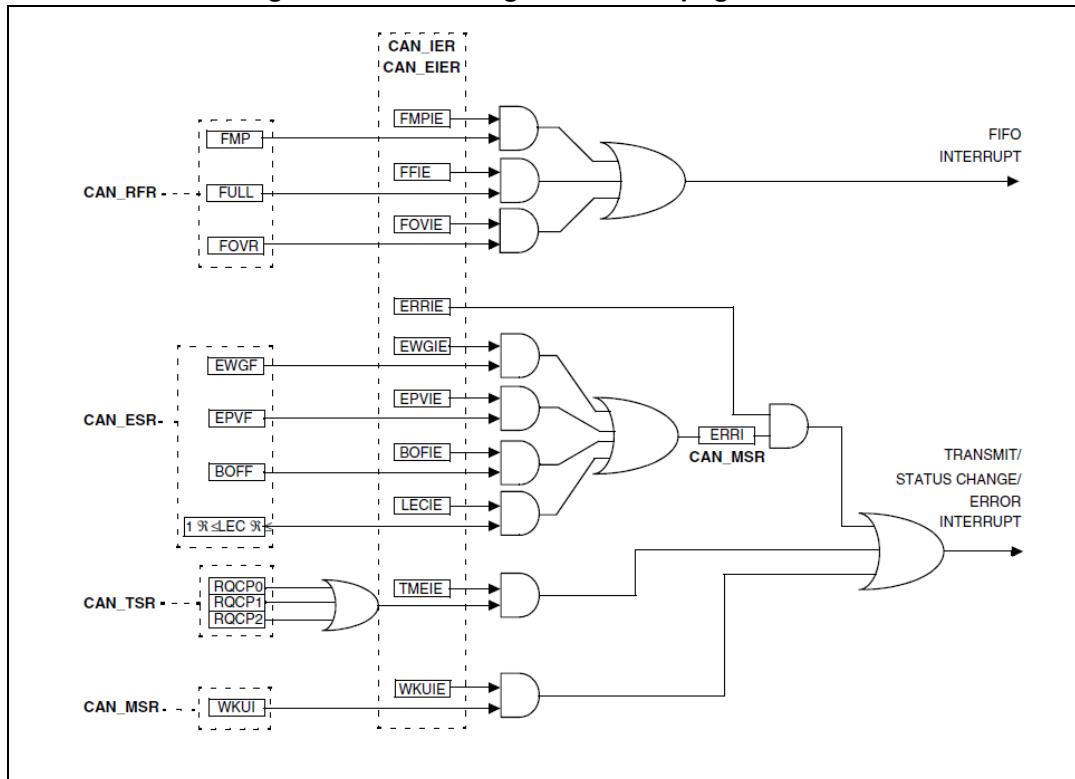
- Legend:  
 $0 \leq N \leq 8$

SOF = Start Of Frame; ID = Identifier; RTR = Remote Transmission Request; IDE = Identifier Extension;  
r0, r1 = Reserved bits; DLC = Data Length Code; CRC = Cyclic Redundancy Code;  
Error flag: 6 dominant bits if node is error active else 6 recessive bits.  
Suspend transmission: applies to error passive nodes only.  
EOF = End of Frame  
ACK = Acknowledge bit (send as recessive)  
Ctrl = Control

## 23.7 Interrupts

Two interrupt vectors are dedicated to beCAN. Each interrupt source can be independently enabled or disabled by means of the CAN Interrupt Enable Register (CAN\_IER) and CAN Error Interrupt Enable register (CAN\_EIER).

**Figure 156. Event flags and interrupt generation**



- The **FIFO interrupt** can be generated by the following events:
  - Reception of a new message, FMP bits in the CAN\_RFR register incremented.
  - FIFO full condition, FULL bit in the CAN\_RFR register set.
  - FIFO overrun condition, FOVR bit in the CAN\_RFR register set.
- The **transmit, error and status change interrupt** can be generated by the following events:
  - Transmit mailbox 0 becomes empty, RQCP0 bit in the CAN\_TSR register set.
  - Transmit mailbox 1 becomes empty, RQCP1 bit in the CAN\_TSR register set.
  - Transmit mailbox 2 becomes empty, RQCP2 bit in the CAN\_TSR register set.
  - Error condition, for more details on error conditions please refer to the CAN Error Status register (CAN\_ESR).
  - Wakeup condition, SOF monitored on the CAN Rx signal.

## 23.8 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore the following registers can be modified by software only while the hardware is in initialization mode:

CAN\_BTR1, CAN\_BTR2, CAN\_FCR1, CAN\_FCR2, CAN\_FMR1, CAN\_FMR2 and CAN\_DGR registers.

Although the transmission of incorrect data will not cause problems at the CAN network level, it can severely disturb the application. A transmit mailbox can be only modified by software while it is in empty state, refer to [Figure 146: Transmit mailbox states](#).

The filters must be deactivated before their value can be modified by software. The modification of the filter configuration (scale or mode) can be done by software only in initialization mode.

## 23.9 Clock system

The clock tolerance limit as specified in CAN protocol is 1.58 % at speeds of up to 125 Kbps. For higher baud rates, it is suggested to use a crystal oscillator.

**Note:** *If the clock security system feature is enabled in the CLK controller (Refer to the description of the CSSEN bit in the [Clock security system register \(CLK\\_CSSR\) on page 97](#)), there is a way to put CAN automatically into the recessive state when a main clock failure occurs, so that the CAN network does not get stuck by the device. However to ensure this, the PG0 I/O pin must be configured in pull-up mode prior to using the beCAN. In this way, when a failure occurs and the I/O alternate function is disabled, the line is pulled-up instead of floating.*

## 23.10 beCAN low power modes

Table 68. beCAN behavior in low power modes

Mode	Description
Wait	No effect on beCAN, except that accesses to Tx/Rx mailboxes and filter values are not possible (CPU clock is stopped). beCAN interrupts cause the device to exit from WAIT mode.
Slow	No effect on beCAN.
Halt/ Active-halt	BeCAN is halted. If the beCAN has been successfully put in Sleep mode (refer to <a href="#">Section 23.4.3</a> ) before executing the halt instruction, any falling edge detected on CAN_RX pin will trigger a beCAN Rx interrupt and wake-up the device from Halt/Active-halt mode.

**Note:** *If a CAN frame is received in Wait, Halt or Active-halt modes, the microcontroller will be woken-up but the CAN frame will be lost.*

## 23.11 beCAN registers

### 23.11.1 CAN master control register (CAN\_MCR)

Address offset: 0x00

Reset value: 0x02

7	6	5	4	3	2	1	0
TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
rw	rw						

#### Bit 7 **TTCM** Time Triggered Communication Mode

0: Time Triggered Communication mode disabled.

1: Time Triggered Communication mode enabled

*Note: For more information on Time Triggered Communication mode, please refer to [Section 23.4.4: Time triggered communication mode](#).*

#### Bit 6 **ABOM** Automatic Bus-Off Management

This bit controls the behavior of the CAN hardware on leaving the Bus-Off state.

0: The Bus-Off state is exited on software request.

Refer to [Section 23.6.5: Error management](#), Bus-Off recovery.

1: The Bus-Off state is exited automatically by hardware once 128 x 11 recessive bits have been monitored.

*Note: For detailed information on the Bus-Off state please refer to [Section 23.6.5: Error management](#).*

#### Bit 5 **AWUM** Automatic wakeup Mode

This bit controls the behavior of the CAN hardware when a message is received in Sleep mode.

0: The sleep mode is exited on software request by clearing the SLEEP bit in the CAN\_MCR register.

1: The sleep mode is exited automatically by hardware when a CAN message is detected. In this case, the SLEEP bit of the CAN\_MCR register and the SLAK bit of the CAN\_MSR register are cleared by hardware, while the WKUI bit of the CAN\_MSR register is set. If the CAN bus is active when the Sleep automatic wakeup request occurs, it does not enter Sleep mode.

**The AWUM and the SLEEP bits must be set simultaneously.**

#### Bit 4 **NART** No Automatic Retransmission

0: The CAN hardware will automatically retransmit the message until it has been successfully transmitted according to the CAN standard.

1: A message will be transmitted only once, independently of the transmission result (successful, error or arbitration lost).

#### Bit 3 **RFLM** Receive FIFO Locked Mode

0: Receive FIFO not locked on overrun. Once a receive FIFO is full the next incoming message will overwrite the previous one.

1: Receive FIFO locked against overrun. Once a receive FIFO is full the next incoming message will be discarded.

**Bit 2 TXFP Transmit FIFO Priority**

This bit controls the transmission order when several mailboxes are pending at the same time.

0: Priority driven by the identifier of the message

1: Priority driven by the request order (chronologically)

**Bit 1 SLEEP Sleep Mode Request**

This bit must be set by software to request the CAN hardware to enter Sleep mode. If the AWUM bit is not set, the Sleep mode is entered as soon as the current CAN activity (CAN frame transmission or reception) has completed. If the AWUM bit is set and the CAN bus is active, the CAN does not enter Sleep mode, the SLEEP bit is not set, and the WKUI bit of the CAN\_MSR register is set.

This bit must be cleared by software to exit Sleep mode. It can be cleared by hardware when the AWUM bit is set and a SOF bit is detected on the CAN Rx signal.

After a reset, the CAN is in Sleep mode and the SLEEP bit is set.

**Bit 0 INRQ Initialization Request**

The software clears this bit to switch the hardware into normal mode. Once 11 consecutive recessive bits have been monitored on the Rx signal the CAN hardware is synchronized and ready for transmission and reception. Hardware signals this event by clearing the INAK bit in the CAN\_MSR register.

Software sets this bit to request the CAN hardware to enter initialization mode. Once software has set the INRQ bit, the CAN hardware waits until the current CAN activity (transmission or reception) is completed before entering the initialization mode. Hardware signals this event by setting the INAK bit in the CAN\_MSR register.

### 23.11.2 CAN master status register (CAN\_MSR)

Address offset: 0x01

Reset value: 0x02

7	6	5	4	3	2	1	0
Reserved	RX	TX	WKUI	ERRI	SLAK	INAK	
r	r	r	rc_w1	rc_w1	r	r	

Bits 7:6 Reserved.

**Bit 5 RX Receive**

1: The CAN hardware is currently receiver.

**Bit 4 TX Transmit**

1: The CAN hardware is currently transmitter.

**Bit 3 WKUI Wakeup Interrupt**

This bit is set by hardware to signal that a SOF bit has been detected while the CAN hardware was in sleep mode. Setting this bit generates a status change interrupt if the WKUIE bit in the CAN\_IER register is set.

This bit is cleared by software writing 1.

**Bit 2 ERRI Error Interrupt**

This bit is set by hardware when a bit of the CAN\_ESR has been set on error detection and the corresponding interrupt in the CAN\_EIER is enabled. Setting this bit generates a status change interrupt if the ERRIE bit in the CAN\_EIER register is set.

This bit is cleared by software writing 1.

**Bit 1 SLAK Sleep Acknowledge**

This bit is set by hardware and indicates to the software that the CAN hardware is now in sleep mode. This bit acknowledges the sleep mode request from the software (set SLEEP bit in CAN\_MCR register).

This bit is cleared by hardware when the CAN hardware has exited Sleep mode. Sleep mode is exited when the SLEEP bit in the CAN\_MCR register is cleared. Please refer to the AWUM bit of the CAN\_MCR register description for detailed information for clearing SLEEP bit.

**Bit 0 INAK Initialization Acknowledge**

This bit is set by hardware and indicates to the software that the CAN hardware is now in initialization mode. This bit acknowledges the initialization request from the software (set INRQ bit in CAN\_MCR register).

This bit is cleared by hardware when the CAN hardware has exited initialization mode and is now synchronized on the CAN bus. To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

### 23.11.3 CAN transmit status register (CAN\_TSR)

Address offset: 0x02

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TXOK2	TXOK1	TXOK0	Reserved	RQCP2	RQCP1	RQCP0
r	r	r	r	r	rc_w1	rc_w1	rc_w1

Bit 7 Reserved.

**Bit 6 TXOK2 Transmission OK for mailbox 2**

This bit is set by hardware when the transmission request on mailbox 2 has been completed successfully. Please refer to [Figure 146](#).

This bit is cleared by hardware when mailbox 2 is requested for transmission or when the software clears the RQCP2 bit.

**Bit 5 TXOK1 Transmission OK for mailbox 1**

This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Please refer to [Figure 146](#).

This bit is cleared by hardware when mailbox 1 is requested for transmission or when the software clears the RQCP1 bit.

**Bit 4 TXOK0 Transmission OK for mailbox 0**

This bit is set by hardware when the transmission request on mailbox 0 has been completed successfully. Please refer to [Figure 146](#).

This bit is cleared by hardware when mailbox 0 is requested for transmission or when the software clears the RQCP0 bit.

Bit 3 Reserved.

Bit 2 **RQCP2** Request Completed for Mailbox 2

This bit is set by hardware to signal that the last request for mailbox 2 has been completed. The request could be a transmit or an abort request.

This bit is cleared by software writing 1.

Bit 1 **RQCP1** Request Completed for Mailbox 1

This bit is set by hardware to signal that the last request for mailbox 1 has been completed. The request could be a transmit or an abort request.

This bit is cleared by software writing 1.

Bit 0 **RQCP0** Request Completed for Mailbox 0

This bit is set by hardware to signal that the last request for mailbox 0 has been completed. The request could be a transmit or an abort request.

This bit is cleared by software writing 1.

**23.11.4 CAN transmit priority register (CAN\_TPR)**

Address offset: 0x03

Reset value: 0x0C

7	6	5	4	3	2	1	0
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE1	CODE0
r	r	r	r	r	r	r	r

Bit 7 **LOW2** Lowest Priority Flag for Mailbox 2

This bit is set by hardware when more than one mailbox is pending for transmission and mailbox 2 has the lowest priority.

*Note: It is set to zero when only one mailbox is pending.*

Bit 6 **LOW1** Lowest Priority Flag for Mailbox 1

This bit is set by hardware when more than one mailbox is pending for transmission and mailbox 1 has the lowest priority.

*Note: It is set to zero when only one mailbox is pending.*

Bit 5 **LOW0** Lowest Priority Flag for Mailbox 0

This bit is set by hardware when more than one mailbox is pending for transmission and mailbox 0 has the lowest priority.

*Note: It is set to zero when only one mailbox is pending.*

Bit 4 **TME2** Transmit Mailbox 2 Empty

This bit is set by hardware when no transmit request is pending for mailbox 2.

*Note: This bit is reserved, forced to 0 by hardware in ST7 beCAN compatibility mode (TXM2E bit = 0 in CAN\_DGR register).*

**Bit 3 TME1** Transmit Mailbox 1 Empty

This bit is set by hardware when no transmit request is pending for mailbox 1.

**Bit 2 TME0** Transmit Mailbox 0 Empty

This bit is set by hardware when no transmit request is pending for mailbox 0.

**Bits 1:0 CODE[1:0]** Mailbox Code

When at least one transmit mailbox is free, this field contains the number of the next free transmit mailbox.

When all transmit mailboxes are pending, this field contains the number of the transmit mailbox with the lowest priority.

*Note:* CODE1 is always 0 in ST7 beCAN compatibility mode (TXM2E bit = 0 in CAN\_DGR register).

### 23.11.5 CAN receive FIFO register (CAN\_RFR)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	RFOM	FOVR	FULL	Reserved	FMP[1:0]		
r	rs	rc_w1	rc_w1	r	r	r	r

Bit 7:6 Reserved.

#### Bit 5 RFOM Release FIFO Output Mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If more than one message is pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

#### Bit 4 FOVR FIFO Overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software writing '1'.

#### Bit 3 FULL FIFO Full

Set by hardware when three messages are stored in the FIFO.

This bit can be cleared by software writing '1' or by releasing the FIFO by means of RFOM.

Bit 2 Reserved.

#### Bits 1:0 FMP[1:0] FIFO Message Pending

These bits indicate how many messages are pending in the receive FIFO.

FMP is increased each time the hardware stores a new message in to the FIFO. FMP is decreased each time the FIFO output mailbox has been released by hardware (RFOM bit has been cleared after prior setting by software).

### 23.11.6 CAN interrupt enable register (CAN\_IER)

Address offset: 0x05

Reset value: 0x00

7	6	5	4	3	2	1	0
WKUIE		Reserved		FOVIE	FFIE	FMPIE	TMEIE
rw		r		rw	rw	rw	rw

Bit 7 **WKUIE** Wakeup Interrupt Enable

0: No interrupt when WKUI is set.

1: Interrupt generated when WKUI bit is set.

Bit 6:4 Reserved.

Bit 3 **FOVIE** FIFO Overrun Interrupt Enable

0: No interrupt when FOVR bit is set.

1: Interrupt generated when FOVR bit is set.

Bit 2 **FFIE** FIFO Full Interrupt Enable

0: No interrupt when FULL bit is set.

1: Interrupt generated when FULL bit is set.

Bit 1 **FMPIE** FIFO Message Pending Interrupt Enable

0: No interrupt on FMP[1:0] bits transition from 0b00 to 0b01.

1: Interrupt generated on FMP[1:0] bits transition from 0b00 to 0b01.

Bit 0 **TMEIE** Transmit Mailbox Empty Interrupt Enable

0: No interrupt when RQCPx bit is set.

1: Interrupt generated when RQCPx bit is set.

### 23.11.7 CAN diagnostic register (CAN\_DGR)

Address offset: 0x06

Reset value: 0x0C

7	6	5	4	3	2	1	0
	Reserved		TXM2E	RX	SAMP	SILM	LBKM
	r		rw	r	r	rw	rw

Bit 7:5 Reserved.

Bit 4 **TXM2E** TX Mailbox 2 enable

0: Force compatibility with ST7 beCAN (2 TX Mailboxes) - reset value

1: Enables the third TX Mailbox (Mailbox number 2)

Bit 3 **RX** CAN Rx Signal

Monitors the actual value of the **CAN\_RX** Pin.

Bit 2 **SAMP** Last sample point

The value of the last sample point.

Bit 1 **SILM** Silent mode

0: Normal operation

1: Silent mode

Bit 0 **LBKM** Loop back mode

0: Loop back mode disabled

1: Loop back mode enabled

### 23.11.8 CAN page select register (CAN\_PSR)

Address offset: 0x07

Reset value: 0x00

7	6	5	4	3	2	1	0		
	Reserved			<b>PS[2:0]</b>					
		r			rw	rw	rw		

Bits 7:3 Reserved.

Bits 2:0 **PS[2:0]** Page select

This register is used to select the register page.

000: Tx Mailbox 0

001: Tx Mailbox 1

010: Acceptance Filter 0:1

011: Acceptance Filter 2:3

100: Acceptance Filter 4:5

101: Tx Mailbox 2

110: Configuration/Diagnostic

111: Receive FIFO

Refer to [Figure 158](#) for more details.

### 23.11.9 CAN error status register (CAN\_ESR)

Address offset: See [Table 71](#).

Reset value: 0000 0000 (00h)

7	6	5	4	3	2	1	0
Reserved	LEC[2:0]			Reserved	BOFF	EPVF	EWGF

Bit 7 Reserved.

#### Bit 6:4 LEC[2:0] Last error code

This field holds a code which indicates the type of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to '0'. The code 7 is unused and may be written by the CPU to check for update.

000: No Error

001: Stuff Error

010: Form Error

011: Acknowledgment Error

100: Bit recessive Error

101: Bit dominant Error

110: CRC Error

111: Set by software

Bit 3 Reserved.

#### Bit 2 BOFF Bus-off flag

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on CAN\_TECR overrun, TEC greater than 255, refer to [Section 23.6.5 on page 396](#).

#### Bit 1 EPVF Error passive flag

This bit is set by hardware when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter greater than 127).

#### Bit 0 EWGF Error warning flag

This bit is set by hardware when the warning limit has been reached. Receive Error Counter or Transmit Error Counter greater than 96.

### 23.11.10 CAN error interrupt enable register (CAN\_EIER)

Address offset: See [Table 71](#).

Reset value: 0000 0000 (00h)

	7	6	5	4	3	2	1	0
	ERRIE	Reserved	LECIE	Reserved	BOFIE	EPVIE	EWGIE	
	rw	r	rw	r	rw	rw	rw	

Bit 7 **ERRIE** Error interrupt enable

0: No interrupt is generated when an error condition is pending in the CAN\_ESR (ERRI bit in CAN\_MSR is set).

1: An interrupt is generated when an error condition is pending in the CAN\_ESR (ERRI bit in CAN\_MSR is set).

Refer to [Figure 156](#) for more details.

Bit 6:5 Reserved.

Bit 4 **LECIE** Last error code interrupt enable

0: ERRI bit is not set when the error code in LEC[2:0] is set by hardware on error detection.

1: ERRI bit is set when the error code in LEC[2:0] is set by hardware on error detection.

Bit 3 Reserved.

Bit 2 **BOFIE** Bus-Off interrupt enable

0: ERRI bit is not set when BOFF is set.

1: ERRI bit is set when BOFF is set.

Bit 1 **EPVIE** Error passive interrupt enable

0: ERRI bit is not set when EPVF is set.

1: ERRI bit is set when EPVF is set.

Bit 0 **EWGIE** Error warning interrupt enable

0: ERRI bit is not set when EWGF is set.

1: ERRI bit is set when EWGF is set.

### 23.11.11 CAN transmit error counter register (CAN\_TECR)

Address offset: See [Table 71](#).

Reset value: 0000 0000 (00h)

	7	6	5	4	3	2	1	0
	TEC[7:0]							
	r	r	r	r	r	r	r	r

Bits 7:0 **TEC[7:0]** Transmit error counter

In case of an error during transmission, this counter is incremented by 8 depending on the error condition as defined by the CAN standard. After every successful transmission the counter is decremented by 1 or reset to 0 if the CAN controller exited from bus-off to error active state. When the counter value exceeds 127, the CAN controller enters the error passive state. When the counter value exceeds 255, the CAN controller enters the bus-off state.

### 23.11.12 CAN receive error counter register (CAN\_RECRR)

Address offset: See [Table 71](#).

Reset value: 0000 0000 (00h)

7	6	5	4	3	2	1	0
REC[7:0]							
r	r	r	r	r	r	r	r

Bits 7:0 **REC[7:0]** Receive error counter

This is the Receive Error Counter implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

### 23.11.13 CAN bit timing register 1 (CAN\_BTR1)

Address offset: See [Table 71](#).

Reset value: 0100 0000 (40h)

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

This register can only be accessed by the software when the CAN hardware is in initialization mode.

Bits 7:6 **SJW[1:0]** Resynchronization jump width

These bits define the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform the resynchronization. Resynchronization Jump Width = (SJW+1).

Bits 5:0 **BRP[5:0]** Baud rate prescaler

These bits define the length of a time quantum.

$$t_q = (BRP+1)/f_{MASTER}$$

### 23.11.14 CAN bit timing register 2 (CAN\_BTR2)

Address offset: See [Table 71](#).

Reset value: 0x23

7	6	5	4	3	2	1	0
Reserved	BS2[2:0]				BS1[3:0]		
r	rw	rw	rw	rw	rw	rw	rw

This register can only be accessed by the software when the CAN hardware is in initialization mode.

Bit 7 Reserved, must be kept cleared.

Bits 6:4 **BS2[2:0]** Bit Segment 2

These bits define the number of time quanta in Bit Segment 2.

Bit Segment 2 = (BS2+1)

Bits 3:0 **BS1[3:0]** Bit Segment 1

These bits define the number of time quanta in Bit Segment 1

Bit Segment 1 = (BS1+1)

For more information on bit timing, please refer to [Section 23.6.6: Bit timing](#).

### 23.11.15 Mailbox registers

This chapter describes the registers of the transmit and receive mailboxes. Refer to [Section 23.6.4: Message storage](#) for detailed register mapping.

Transmit and receive mailboxes have the same registers except:

- CAN\_MCSR register in a transmit mailbox is replaced by CAN\_MFMIR register in a receive mailbox.
- A receive mailbox is always write protected.
- A transmit mailbox is write enabled only while empty (the corresponding TME bit in the CAN\_TPR register is set).

**Caution:** As the mailbox registers usually have no defined reset value, the user should not rely on the initial setup and should always fill all the configuration bits accordingly.

#### CAN message control/status register (CAN\_MCSR)

Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	TERR	ALST	TXOK	RQCP	ABRQ	TXRQ	
r	r	r	r	rc_w1	rs	rs	

**Note:** *This register is implemented only in transmit mailboxes. In receive mailboxes, the CAN\_MFMIR register is mapped at this location.*

Bits 7:6 Reserved.

##### Bit 5 **TERR** Transmission error

This bit is updated by hardware after each transmission attempt.

0: The previous transmission was successful

1: The previous transmission failed due to an error

##### Bit 4 **ALST** Arbitration lost

This bit is updated by hardware after each transmission attempt.

0: The previous transmission was successful

1: The previous transmission failed due to an arbitration lost

##### Bit 3 **TXOK** Transmission OK

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

**Note:** *This bit has the same value as the corresponding TXOKx bit in the CAN\_TSR register.*

**Bit 2 RQCP Request completed**

Set by hardware when the last request (transmit or abort) has been performed.

Cleared by software writing a “1” or by hardware on transmission request.

*Note: This bit has the same value as the corresponding RQCPx bit of the CAN\_TSR register.*

*Clearing this bit clears all the status bits (TXOK, ALST and TERR) in the CAN\_MCSR register and the corresponding RQCPx and TXOKx bits in the CAN\_TSR register.*

**Bit 1 ABRQ Abort request for mailbox**

Set by software to abort the transmission request for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

Setting this bit has no effect when the mailbox is not pending for transmission.

**Bit 0 TXRQ Transmit mailbox request**

Set by software to request the transmission for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

**CAN mailbox filter match index register (CAN\_MFMIR)**

Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

7	6	5	4	3	2	1	0
FMI[7:0]							
r	r	r	r	r	r	r	r

*Note: This register is implemented only in receive mailboxes. In transmit mailboxes, the CAN\_MCSR register is mapped at this location.*

**Bits 7:0 FMI[7:0] Filter match index**

This register contains the index of the filter the message stored in the mailbox passed through. For more details on identifier filtering please refer to [Section 23.6.3: Identifier filtering - Filter Match Index](#) paragraph.

### CAN mailbox identifier register 1 (CAN\_MIDR1)

Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0xXX

7	6	5	4	3	2	1	0		
Reserved	IDE	RTR	STID[10:6] / EXID[28:24]						
r	rw	rw	rw	rw	rw	rw	rw		

Bit 7 Reserved.

Bit 6 **IDE** Extended identifier

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 5 **RTR** Remote transmission request

0: Data frame

1: Remote frame

Bits 4:0 **STID[10:6]** Standard identifier

5 most significant bits of the standard part of the identifier.

or

**EXID[28:24]** Extended identifier

5 most significant bits of the “Base” part of extended identifier.

### CAN mailbox identifier register 2 (CAN\_MIDR2)

Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0xXX

7	6	5	4	3	2	1	0
STID[5:0] / EXID[23:18]						EXID[17:16]	
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:2 **STID[5:0]** Standard Identifier

6 least significant bits of the standard part of the identifier.

or

**EXID[23:18]** Extended Identifier

6 least significant bits of the “Base” part of extended identifier.

Bits 1:0 **EXID[17:16]** Extended Identifier

2 most significant bits of the “Extended” part of the extended identifier.

**CAN mailbox identifier register 3 (CAN\_MIDR3)**Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

7	6	5	4	3	2	1	0
EXID[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **EXID[15:8]** Extended identifier

Bit 15 to 8 of the “Extended” part of the extended identifier.

**CAN mailbox identifier register 4 (CAN\_MIDR4)**Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

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

Bits 7:0 **EXID[7:0]** Extended identifier

8 least significant bits of the “Extended” part of the extended identifier.

**CAN mailbox data length control register (CAN\_MDLCR)**Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

7	6	5	4	3	2	1	0		
TGT		Reserved		DLC[3:0]					
rw		r		rw	rw	rw	rw		

Bit 7 **TGT** Transmit global time

This bit should be used only when the hardware is in the Time Trigger Communication mode, TTCTM bit in the CAN\_MCR register is set. It must be cleared by user in Normal mode to transfer last two data bytes correctly

0: CAN\_MTSRH and CAN\_MTSRL registers are not sent.

1: CAN\_MTSRH and CAN\_MTSRL registers are sent in the last two data bytes of the message.

Bits 6:4 Reserved.

Bits 3:0 **DLC[3:0]** Data length code

This field defines the number of data bytes in a data frame or a remote frame request.

**CAN mailbox data register x (CAN\_M DAR) (x= 1 .. 8)**Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

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

**Bits 7:0 DATA[7:0] Data**

A data byte of the message. A message can contain from 0 to 8 data bytes.

*Note:* These bits are write protected when the mailbox is not in empty state.**CAN mailbox time stamp register low (CAN\_MTSRL)**Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

7	6	5	4	3	2	1	0
TIME[7:0]							
r	r	r	r	r	r	r	r

**Bits 7:0 TIME[7:0] Message time stamp low**

This field contains the low byte of the 16-bit timer value captured at the SOF detection.

**CAN mailbox time stamp register high (CAN\_MTSRH)**Address offset: See [Table 66.](#) and [Table 67.](#)

Reset value: 0XX

7	6	5	4	3	2	1	0
TIME[15:8]							
r	r	r	r	r	r	r	r

**Bits 7:0 TIME[15:8] Message time stamp high**

This field contains the high byte of the 16-bit timer value captured at the SOF detection.

### 23.11.16 CAN filter registers

#### CAN filter mode register 1 (CAN\_FMR1)

Address offset: See [Table 71](#).

Reset value: 0x00

7	6	5	4	3	2	1	0
FMH3	FML3	FMH2	FML2	FMH1	FML1	FMH0	FML0
rw							

##### Bit 7 **FMH3** Filter 3 mode high

Mode of the high identifier/mask registers of Filter 3.

0: High registers are in mask mode

1: High registers are in identifier list mode

##### Bit 6 **FML3** Filter 3 mode low

Mode of the low identifier/mask registers of Filter 3.

0: Low registers are in mask mode

1: Low registers are in identifier list mode

##### Bit 5 **FMH2** Filter 2 mode high

Mode of the high identifier/mask registers of Filter 2.

0: High registers are in mask mode

1: High registers are in identifier list mode

##### Bit 4 **FML2** Filter 2 mode low

Mode of the low identifier/mask registers of Filter 2.

0: Low registers are in mask mode

1: Low registers are in identifier list mode

##### Bit 3 **FMH1** Filter 1 mode high

Mode of the high identifier/mask registers of Filter 1.

0: High registers are in mask mode

1: High registers are in identifier list mode

##### Bits 2 **FML1** Filter 1 mode low

Mode of the low identifier/mask registers of filter 1.

0: Low registers are in mask mode

1: Low registers are in identifier list mode

##### Bit 1 **FMH0** Filter 0 mode high

Mode of the high identifier/mask registers of filter 0.

0: High registers are in mask mode

1: High registers are in identifier list mode

##### Bit 0 **FML0** Filter 0 mode low

Mode of the low identifier/mask registers of filter 0.

0: Low registers are in mask mode

1: Low registers are in identifier list mode

**CAN filter mode register 2 (CAN\_FMR2)**Address offset: See [Table 71](#).

Reset value: 0x00

7	6	5	4	3	2	1	0
		Reserved		FMH5	FML5	FMH4	FML4
		r		rw	rw	rw	rw

Bits 7:4 Reserved.

**Bit 3 FMH5** Filter 5 mode high

Mode of the high identifier/mask registers of Filter 5.

0: High registers are in mask mode

1: High registers are in identifier list mode

**Bits 2 FML5** Filter 5 mode low

Mode of the low identifier/mask registers of filter 5.

0: Low registers are in mask mode

1: Low registers are in identifier list mode

**Bit 1 FMH4** Filter 4 mode high

Mode of the high identifier/mask registers of filter 4.

0: High registers are in mask mode

1: High registers are in identifier list mode

**Bit 0 FML4** Filter 4 mode low

Mode of the low identifier/mask registers of filter 4.

0: Low registers are in mask mode

1: Low registers are in identifier list mode

**CAN filter configuration register 1 (CAN\_FCR1)**

Address offset: See [Table 71](#).

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	FSC11	FSC10	FACT1	Reserved	FSC01	FSC00	FACT0
r	rw	rw	rw	r	rw	rw	rw

Bit 7 Reserved.

Bits 6:5 **FSC1[1:0]** Filter scale configuration

These bits define the scale configuration of Filter 1.

Bit 4 **FACT1** Filter Active

The software sets this bit to activate Filter 1. To modify the Filter 1 registers (CAN\_F1Rx), the FACT1 bit must be cleared.

0: Filter 1 is not active

1: Filter 1 is active

Bit 3 Reserved.

Bits 2:1 **FSC0[1:0]** Filter scale configuration

These bits define the scale configuration of Filter 0.

Bit 0 **FACT0** Filter active

The software sets this bit to activate Filter 0. To modify the Filter 0 registers (CAN\_F0Rx), the FACT0 bit must be cleared.

0: Filter 0 is not active

1: Filter 0 is active

### CAN filter configuration register 2 (CAN\_FCR2)

Address offset: See [Table 71](#).

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	FSC31	FSC30	FACT3	Reserved	FSC21	FSC20	FACT2
r	rw	rw	rw	r	rw	rw	rw

Bit 7 Reserved.

Bits 6:5 **FSC3[1:0]** Filter scale configuration

These bits define the scale configuration of Filter 3.

Bit 4 **FACT3** Filter active

The software sets this bit to activate Filter 3. To modify the Filter 3 registers (CAN\_F3Rx) the FACT3 bit must be cleared.

0: Filter 3 is not active

1: Filter 3 is active

Bit 3 Reserved.

Bits 2:1 **FSC2[1:0]** Filter scale configuration

These bits define the scale configuration of Filter 2.

Bit 0 **FACT2** Filter active

The software sets this bit to activate Filter 2. To modify the Filter 2 registers (CAN\_F2Rx), the FACT2 bit must be cleared.

0: Filter 2 is not active

1: Filter 2 is active

**CAN filter configuration register 3 (CAN\_FCR3)**

Address offset: See [Table 71](#).

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	FSC51	FSC50	FACT5	Reserved	FSC41	FSC40	FACT4
r	rw	rw	rw	r	rw	rw	rw

Bit 7 Reserved.

Bits 6:5 **FSC5[1:0]** Filter scale configuration

These bits define the scale configuration of Filter 5.

Bit 4 **FACT5** Filter active

The software sets this bit to activate Filter 5. To modify the Filter 5 registers (CAN\_F5Rx) the FACT5 bit must be cleared.

0: Filter 5 is not active

1: Filter 5 is active

Bit 3 Reserved.

Bits 2:1 **FSC4[1:0]** Filter scale configuration

These bits define the scale configuration of Filter 4.

Bit 0 **FACT4** Filter active

The software sets this bit to activate Filter 4. To modify the Filter 4 registers (CAN\_F4Rx), the FACT4 bit must be cleared.

0: Filter 4 is not active

1: Filter 4 is active

**CAN filter bank i register x (CAN\_FiRx) (i = 0 .. 5, x = 1 .. 8)**

Address offset: See [Figure 158](#).

Reset value: 0xXX

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

Bits 7:0 **FB[7:0]: Filter bits**

- **Identifier**

Each bit of the register specifies the level of the corresponding bit of the expected identifier.

- 0: Dominant bit is expected
- 1: Recessive bit is expected

- **Mask**

Each bit of the register specifies whether the bit of the associated identifier register must match with the corresponding bit of the expected identifier or not.

- 0: Don't care, the bit is not used for the comparison
- 1: Must match, the bit of the incoming identifier must have the same level has specified in the corresponding identifier register of the filter.

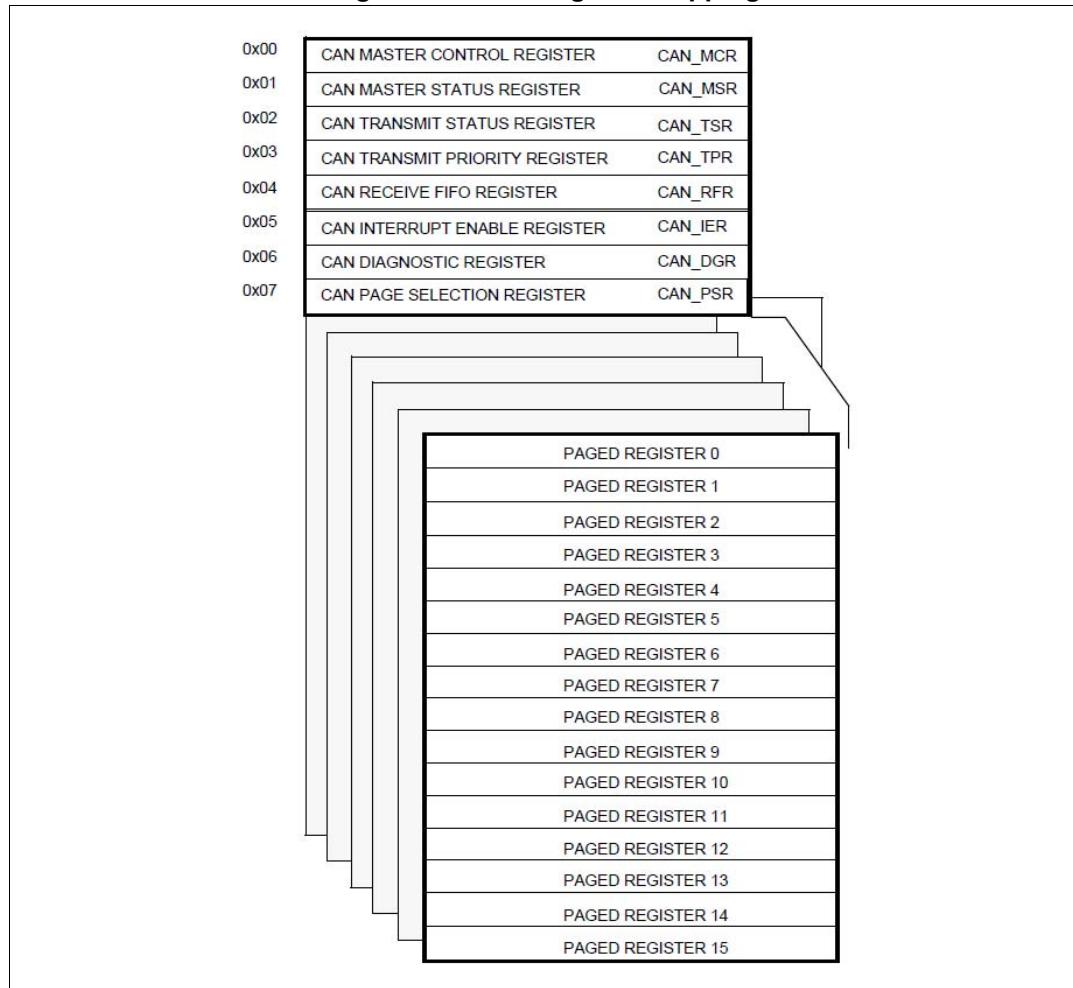
**Note:** *Each filter i is composed of 8 registers, CAN\_FiR1..8. Depending on the scale and mode configuration of the filter the function of each register can differ. For the filter mapping, functions description and mask registers association, refer to Section [Figure 23.6.3: Identifier filtering](#).*

*A Mask/Identifier register in **mask mode** has the same bit mapping as in **identifier list mode**.*

**Note:** *To modify these registers, the corresponding FACT bit in the CAN\_FCRx register must be cleared.*

## 23.12 CAN register map

Figure 157. CAN register mapping



### 23.12.1 Page mapping for CAN

Figure 158. CAN page mapping

	PAGE 0	PAGE 1	PAGE 2	PAGE 3	PAGE 4
0x00	CAN_MCSR	CAN_MCSR	CAN_F0R1	CAN_F2R1	CAN_F4R1
0x01	CAN_MDLCR	CAN_MDLCR	CAN_F0R2	CAN_F2R2	CAN_F4R2
0x02	CAN_MIDR1	CAN_MIDR1	CAN_F0R3	CAN_F2R3	CAN_F4R3
0x03	CAN_MIDR2	CAN_MIDR2	CAN_F0R4	CAN_F2R4	CAN_F4R4
0x04	CAN_MIDR3	CAN_MIDR3	CAN_F0R5	CAN_F2R5	CAN_F4R5
0x05	CAN_MIDR4	CAN_MIDR4	CAN_F0R6	CAN_F2R6	CAN_F4R6
0x06	CAN_MDAR1	CAN_MDAR1	CAN_F0R7	CAN_F2R7	CAN_F4R7
0x07	CAN_MDAR2	CAN_MDAR5	CAN_F0R8	CAN_F2R8	CAN_F4R8
0x08	CAN_MDAR3	CAN_MDAR6	CAN_F1R1	CAN_F3R1	CAN_F5R1
0x09	CAN_MDAR4	CAN_MDAR4	CAN_F1R2	CAN_F3R2	CAN_F5R2
0x0A	CAN_MDAR5	CAN_MDAR5	CAN_F1R3	CAN_F3R3	CAN_F5R3
0x0B	CAN_MDAR6	CAN_MDAR6	CAN_F1R4	CAN_F3R4	CAN_F5R4
0x0C	CAN_MDAR7	CAN_MDAR7	CAN_F1R5	CAN_F3R5	CAN_F5R5
0x0D	CAN_MDAR8	CAN_MDAR8	CAN_F1R6	CAN_F3R6	CAN_F5R6
0x0E	CAN_MTSRL	CAN_MTSRL	CAN_F1R7	CAN_F3R7	CAN_F5R7
0x0F	CAN_MTSRH	CAN_MTSRH	CAN_F1R8	CAN_F3R8	CAN_F5R8
	Tx Mailbox 0	Tx Mailbox 1	Acceptance Filter 0:1	Acceptance Filter 2:3	Acceptance Filter 4:5
	PAGE 5	PAGE 6	PAGE 7		
0x00	CAN_MCSR	CAN_ESR	CAN_MFMIR		
0x01	CAN_MDLCR	CAN_EIER	CAN_MDLCR		
0x02	CAN_MIDR1	CAN_TECR	CAN_MIDR1		
0x03	CAN_MIDR2	CAN_RECRR	CAN_MIDR2		
0x04	CAN_MIDR3	CAN_BTR1	CAN_MIDR3		
0x05	CAN_MIDR4	CAN_BTR2	CAN_MIDR4		
0x06	CAN_MDAR1	Reserved	CAN_MDAR1		
0x07	CAN_MDAR2	Reserved	CAN_MDAR2		
0x08	CAN_MDAR3	CAN_FMR1	CAN_MDAR3		
0x09	CAN_MDAR4	CAN_FMR2	CAN_MDAR4		
0x0A	CAN_MDAR5	CAN_FCR1	CAN_MDAR5		
0x0B	CAN_MDAR6	CAN_FCR2	CAN_MDAR6		
0x0C	CAN_MDAR7	CAN_FCR3	CAN_MDAR7		
0x0D	CAN_MDAR8	Reserved	CAN_MDAR8		
0x0E	CAN_MTSRL	Reserved	CAN_MTSRL		
0x0F	CAN_MTSRH	Reserved	CAN_MTSRH		
	Tx Mailbox 2 (if TXM2E=1 in CAN_DGR register)	Configuration/Diagnostic	Receive FIFO		

**Table 69. beCAN control and status page - register map and reset values**

<b>Address Offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x00	CAN_MCR Reset Value	TTCM 0	ABOM 0	AWUM 0	NART 0	RFLM 0	TXFP 0	SLEEP 1	INRQ 0
0x01	CAN_MSR Reset Value	0	0	RX 0	TX 0	WKUI 0	ERRI 0	SLAK 1	INAK 0
0x02	CAN_TSR Reset Value	0	TXOK2 0	TXOK1 0	TXOK0 0	0	RQCP2 0	RQCP1 0	RQCP0 0
0x03	CAN_TPR Reset Value	LOW2 0	LOW1 0	LOW0 0	TME2 0	TME1 1	TME0 1	CODE1 0	CODE0 0
0x04	CAN_RFR Reset Value	0	0	RFOM 0	FOVR 0	FULL 0	0	FMP1 0	FMP0 0
0x05	CAN_IER Reset Value	WKUIE 0	0	0	0	FOVIE 0	FFIE 0	FMPIE 0	TMEIE 0
0x06	CAN_DGR Reset Value	0	0	0	TXM2E 0	RX 1	SAMP 1	SILM 0	LBKM 0
0x07	CAN_PSR Reset Value	0	0	0	0	0	PS2 0	PS1 0	PS0 0

**Table 70. beCAN mailbox pages - register map and reset values**

<b>Address Offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x00 Receive	CAN_MFMIR Reset Value	FMI7 x	FMI6 x	FMI5 x	FMI4 x	FMI3 x	FMI2 x	FMI1 x	FMI0 x
0x00 Transmit	CAN_MCSR Reset Value	0	0	TERR 0	ALST 0	TXOK 0	RQCP 0	ABRQ 0	TXRQ 0
0x01	CAN_MDLCR Reset Value	TGT x	x	x	x	DLC3 x	DLC2 x	DLC1 x	DLC0 x
0x02	CAN_MIDR1 Reset Value	x	IDE	RTR	STID10 / EXID28 x	STID9 / EXID27 x	STID8 / EXID26 x	STID7 / EXID25 x	STID6 / EXID24 x
0x03	CAN_MIDR2 Reset Value	STID5 / EXID23 x	STID4 / EXID22 x	STID3 / EXID21 x	STID2 / EXID20 x	STID1 / EXID19 x	STID0 / EXID18 x	EXID17 x	EXID16 x
0x04	CAN_MIDR3 Reset Value	EXID15 x	EXID14 x	EXID13 x	EXID12 x	EXID11 x	EXID10 x	EXID9 x	EXID8 x
0x05	CAN_MIDR4 Reset Value	EXID7 x	EXID6 x	EXID5 x	EXID4 x	EXID3 x	EXID2 x	EXID1 x	EXID0 x
0x06:0D	CAN_MDAR1:8 Reset Value	MDAR7 x	MDAR6 x	MDAR5 x	MDAR4 x	MDAR3 x	MDAR2 x	MDAR1 x	MDAR0 x
0x0E	CAN_MTSRL Reset Value	TIME7 x	TIME6 x	TIME5 x	TIME4 x	TIME3 x	TIME2 x	TIME1 x	TIME0 x
0x0F	CAN_MTSRH Reset Value	TIME15 x	TIME14 x	TIME13 x	TIME12 x	TIME11 x	TIME10 x	TIME9 x	TIME8 x

**Table 71. beCAN filter configuration page - register map and reset values**

<b>Address Offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x00	CAN_ESR Reset Value	0	LEC2 0	LEC1 0	LEC0 0	0	BOFF 0	EPVF 0	EWGF 0
0x01	CAN_EIER Reset Value	ERRIE 0	0	0	LECIE 0	0	BOFIE 0	EPVIE 0	EWGIE 0
0x02	CAN_TECR Reset Value	TEC7 0	TEC6 0	TEC5 0	TEC4 0	TEC3 0	TEC2 0	TEC1 0	TEC0 0
0x03	CAN_RECRR Reset Value	REC7 0	REC6 0	REC5 0	REC4 0	REC3 0	REC2 0	REC1 0	REC0 0
0x04	CAN_BTR1 Reset Value	SJW1 0	SJW0 1	BRP5 0	BRP4 0	BRP3 0	BRP2 0	BRP1 0	BRP0 0
0x05	CAN_BTR2 Reset Value	0	BS22 0	BS21 1	BS20 0	BS13 0	BS12 0	BS11 1	BS10 1
0x06	Reserved	X	X	X	X	X	X	X	X
0x07	Reserved	X	X	X	X	X	X	X	X
0x08	CAN_FMR1 Reset Value	FMH3 0	FML3 0	FMH2 0	FML2 0	FMH1 0	FML1 0	FMH0 0	FML0 0
0x09	CAN_FMR2 Reset Value	0	0	0	0	FMH5 0	FML5 0	FMH4 0	FML4 0
0x0A	CAN_FCR1 Reset Value	0	FSC11 0	FSC10 0	FACT1 0	0	FSC01 0	FSC00 0	FACT0 0
0x0B	CAN_FCR2 Reset Value	0	FSC31 0	FSC30 0	FACT3 0	0	FSC21 0	FSC20 0	FACT2 0
0x0C	CAN_FCR3 Reset Value	0	FSC51 0	FSC50 0	FACT5 0	0	FSC41 0	FSC40 0	FACT4 0

## 24 Analog/digital converter (ADC)

### 24.1 Introduction

ADC1 and ADC2 are 10-bit successive approximation Analog to Digital Converters. They have up to 16 multiplexed input channels (the exact number of channels is indicated in the datasheet pin description). A/D Conversion of the various channels can be performed in single, and continuous modes.

ADC1 has extended features for scan mode, buffered continuous mode and analog watchdog. Refer to the datasheet for information about the availability of ADC1 and ADC2 in specific product types.

### 24.2 ADC main features

These features are available in ADC1 and ADC2.

- 10-bit resolution
- Single and continuous conversion modes
- Programmable prescaler:  $f_{\text{MASTER}}$  divided by 2 to 18
- External trigger option using external interrupt (ADC\_ETR) or timer trigger (TRGO)
- Analog zooming (in devices with  $V_{\text{REF}}$  pins)
- Interrupt generation at End of Conversion
- Data alignment with in-built data coherency
- ADC input range:  $V_{\text{SSA}} \leq V_{\text{IN}} \leq V_{\text{DDA}}$

### 24.3 ADC extended features

These features are available in ADC1.

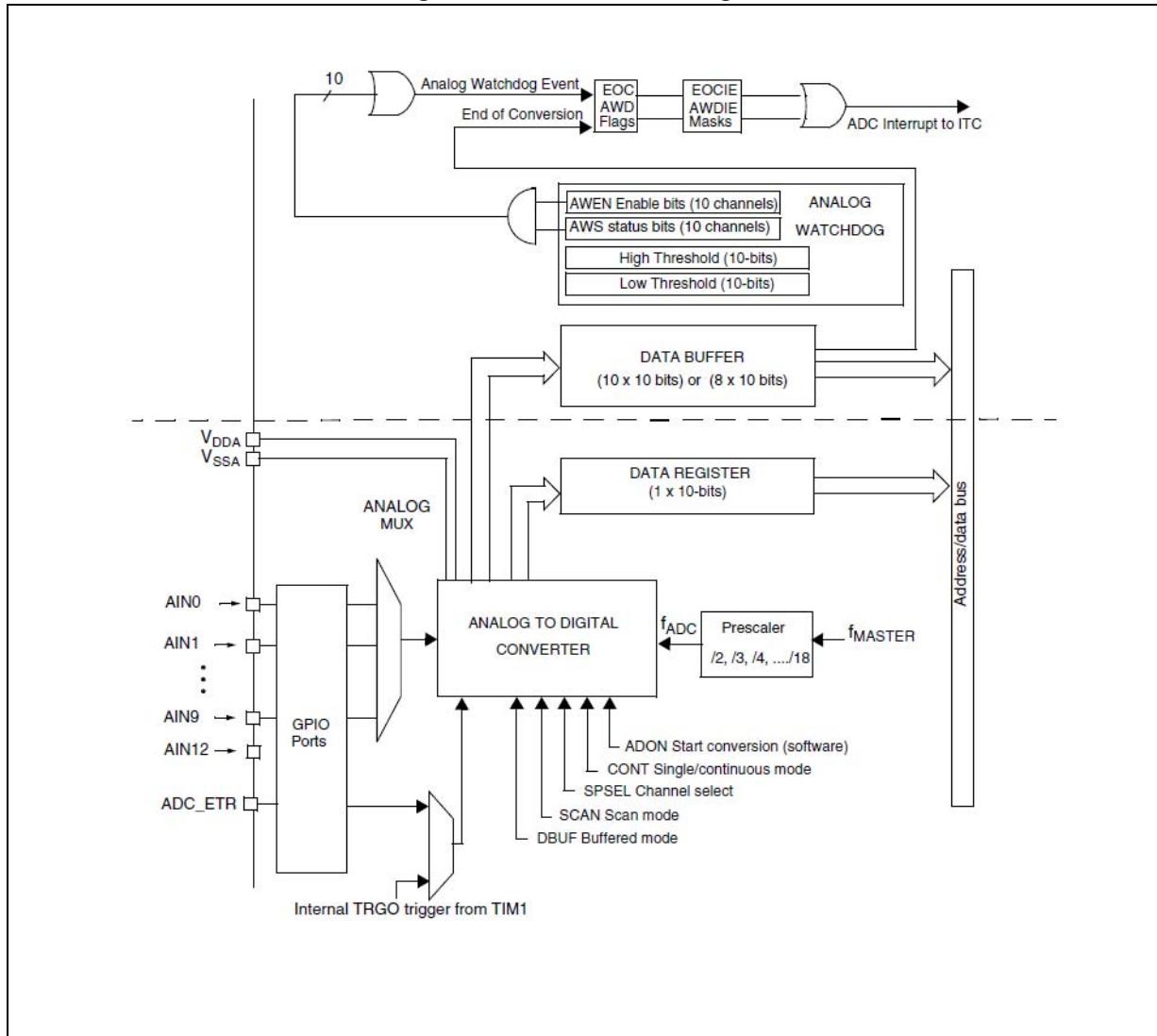
- Buffered continuous conversion mode<sup>(a)</sup>
- Scan mode for single and continuous conversion
- Analog watchdog with upper and lower thresholds
- Interrupt generation at analog watchdog event

The block diagrams of ADC1 and ADC2 are shown in [Figure 159](#) and [Figure 160](#)

---

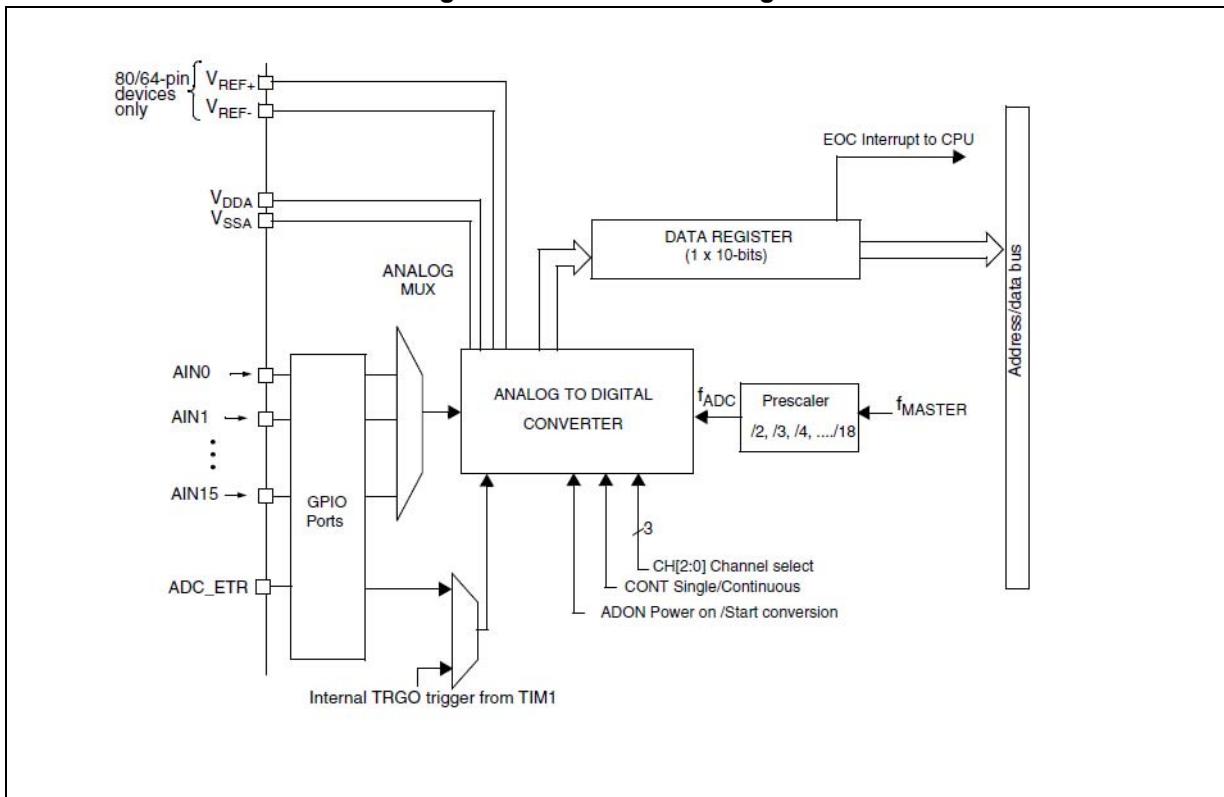
a. Data buffer size is product dependent (10 x 10 bits or 8 x 10 bits). Please refer to the datasheet.

Figure 159. ADC1 block diagram



1. Refer to the product datasheet for AIN12 availability.

Figure 160. ADC2 block diagram



## 24.4 ADC pins

**Table 72. ADC pins**

Name	Signal type	Remarks
$V_{DDA}$	Input, Analog supply	Analog power supply. This input is bonded to $V_{DD}$ in devices that have no external $V_{DDA}$ pin.
$V_{SSA}$	Input, Analog supply ground	Ground for analog power supply. This input is bonded to $V_{SS}$ in devices that have no external $V_{SSA}$ pin.
$V_{REF-}$	Input, Analog Reference negative	The lower/negative reference voltage for the ADC, ranging from $V_{SSA}$ to ( $V_{SSA} + 500$ mV). This input is bonded to $V_{SSA}$ in devices that have no external $V_{REF-}$ pin (packages with 48 pins or less)
$V_{REF+}$	Input, Analog Reference positive	The higher/positive reference voltage for the ADC, ranging from 2.75 V to $V_{DDA}$ . This input is bonded to $V_{DDA}$ in devices that have no external $V_{REF+}$ pin (packages with 48 pins or less)
AIN[15:0]	Analog input signals	Up to 16 analog input channels, which are converted by the ADC one at a time.
ADC_ETR	Digital input signals	External trigger.

## 24.5 ADC functional description

### 24.5.1 ADC on-off control

The ADC can be powered-on by setting the ADON bit in the ADC\_CR1 register. When the ADON bit is set for the first time, it wakes up the ADC from power down mode. To start conversion, set the ADON bit in the ADC\_CR1 register with a second write instruction.

At the end of conversion, the ADC remains powered on and you have to set the ADON bit only once to start the next conversion.

If the ADC is not used for a long time, it is recommended to switch it to power down mode to decrease power consumption. This is done by clearing the ADON bit.

When the ADC is powered on, the digital input and output stages of the selected channel are disabled independently on the GPIO pin configuration. It is therefore recommended to select the analog input channel before powering on the ADC (see [Section 24.5.3: Channel selection](#)).

### 24.5.2 ADC clock

The clock supplied to the ADC can be a prescaled  $f_{MASTER}$  clock. The prescaling factor of the clock depends on the SPSEL[2:0] bits in the ADC\_CR1 register.

### 24.5.3 Channel selection

There are up to 16 external input channels that can be selected through CH[0:3] bits of the ADC\_CSR register. The number of external channels depends on the device (refer to the *product datasheets*).

If the channel selection is changed during a conversion, the current conversion is reset and a new start pulse is sent to the ADC.

### 24.5.4 Conversion modes

The ADC supports five conversion modes: single mode, continuous mode, buffered continuous mode, single scan mode, continuous scan mode.

*Note:* *ADC1 AIN12 channel cannot be selected in ADC scan mode or with analog watchdog. Values converted from AIN12 are stored only in the ADC\_DRH/ADC\_DRL registers. Refer to product datasheet for AIN12 availability.*

#### Single mode

In Single conversion mode, the ADC does one conversion on the channel selected by the CH[3:0] bits in the ADC\_CSR register. This mode is started by setting the ADON bit in the ADC\_CR1 register, while the CONT bit is 0.

Once the conversion is complete, the converted data are stored in the ADC\_DR register, the EOC (End of Conversion) flag is set and an interrupt is generated if the EOCIE bit is set.

#### Continuous and buffered continuous modes

In continuous conversion mode, the ADC starts another conversion as soon as it finishes one. This mode is started by setting the ADON bit in the ADC\_CR1 register, while the CONT bit is set.

- If buffering is not enabled (DBUF bit = 0 in the ADC\_CR3 register), the converted data is stored in the ADC\_DR register and the EOC (End of Conversion) flag is set. An interrupt is generated if the EOCIE bit is set. Then a new conversion starts automatically.
- If buffering is enabled (DBUF bit =1) the data buffer is filled with the results of 8 or 10 consecutive conversions performed on a single channel. When the buffer is full, the EOC (End of Conversion) flag is set and an interrupt is generated if the EOCIE bit is set. Then a new set of 8 or 10 conversions starts automatically. The OVR flag is set if one of the data buffer registers is overwritten before it has been read (see [Section 24.5.5](#)).

To stop continuous conversion, reset the CONT bit to stop conversion or reset the ADON bit to power off the ADC.

#### Single scan mode

This mode is used to convert a sequence of analog channels from AIN0 to AINn where 'n' is the channel number defined by the CH[3:0] bits in the ADC\_CSR register. During the scan conversion sequence the CH[3:0] bits are updated by hardware and contain the channel number currently being converted.

Single scan mode is started by setting the ADON bit while the SCAN bit is set and the CONT bit is cleared.

**Note:** When using scan mode, it is not possible to use channels AIN0 to AINn in output mode because the output stage of each channel is disabled when it is selected by the ADC multiplexer.

A single conversion is performed for each channel starting with AIN0 and the data is stored in the data buffer registers ADC\_DBxR. When the last channel (channel 'n') has been converted, the EOC (End of Conversion) flag is set and an interrupt is generated if the EOCIE bit is set.

The converted values for each channel can be read from the data buffer registers. The OVR flag is set if one of the data buffer registers is overwritten before it has been read (see [Section 24.5.5](#)).

Do not clear the SCAN bit while the conversion sequence is in progress. Single scan mode can be stopped immediately by clearing the ADON bit.

To start a new SCAN conversion, clear the EOC bit and set the ADON bit in the ADC\_CR1 register.

### Continuous scan mode

This mode is like single scan mode except that each time the last channel has been converted, a new scan conversion from channel 0 to channel n starts automatically. The OVR flag is set if one of the data buffer registers is overwritten before it has been read (see [Section 24.5.5](#)).

Continuous scan mode is started by setting the ADON bit while the SCAN and CONT bits are set.

Do not clear the SCAN bit while scan conversion is in progress.

Continuous scan mode can be stopped immediately by clearing the ADON bit. Alternatively if the CONT bit is cleared while conversion is ongoing, conversion stops the next time the last channel has been converted.

**Caution:** In scan mode, do not use a bit manipulation instruction (BRES) to clear the EOC flag. This is because this performs a read-modify-write on the whole ADC\_CSR register, reading the current channel number from the CH[3:0] register and writing it back, which changes the last channel number for the scan sequence.

The correct way to clear the EOC flag in continuous scan mode is to load a byte in the ADC\_CSR register from a RAM variable, clearing the EOC flag and reloading the last channel number for the scan sequence

## 24.5.5 Overrun flag

The OVR error flag is set by hardware in buffered continuous mode, single scan or continuous scan modes. It indicates that one of the ten data buffer registers was overwritten by a new converted value before the previous value was read. In this case, it is recommended to start a new conversion.

**Note:** Setting the ADON bit automatically clears the OVR flag.

### 24.5.6 Analog watchdog

The analog watchdog is enabled by default for single conversion and non-buffered continuous conversion modes.

The AWD analog watchdog flag is set if the analog voltage converted by the ADC is below a low threshold or above a high threshold as shown in [Figure 161](#). These thresholds are programmed in the ADC\_HTR and ADC\_LTR 10-bit registers. An interrupt can be enabled by setting the AWDIE bit in the ADC\_CSR register.

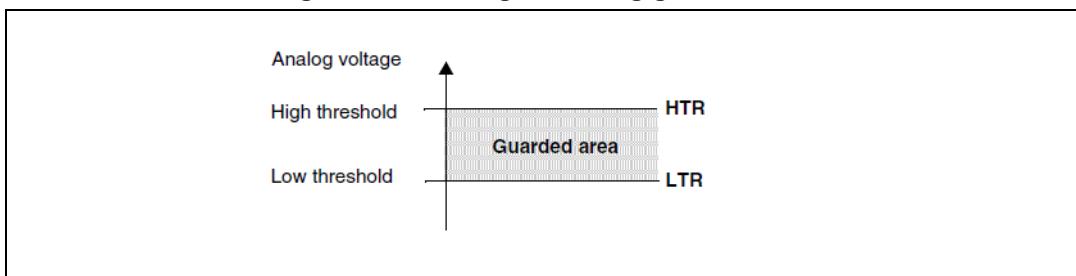
For Scan mode, the analog watchdog can be enabled on selected channels using the AWENx bits in the ADC\_AWCRR and ADC\_AWCRL registers. The watchdog status for each channel is obtained by reading the AWSx bits in the ADC\_AWSRH and ADC\_AWSRL registers. If any of the AWS flags are set, this also sets the AWD flag. Depending on the AWDIE interrupt enable bit, an interrupt is generated at the end of the SCAN sequence. The interrupt routine should then clear the AWS flag and the global AWD flag in the ADC\_CSR register.

For Buffered continuous mode, the analog watchdog can be enabled on selected buffers, and is managed as described for scan mode, with the difference the buffers contain the results of continuous conversions performed on a single channel.

Refer to [Section 24.7](#) for more details on interrupts.

*Note:* To optimize analog watchdog interrupt latency in scan or buffered continuous mode, it recommended to use the last channels in the conversion sequence.

**Figure 161. Analog watchdog guarded area**



#### 24.5.7 Conversion on external trigger

Conversion can be triggered by an rising edge event on the ADC\_ETR pin or a TRGO event from a timer. Refer to the datasheet for details on the timer trigger, as this is product dependent). If the EXTTRIG control bit is set then either of the external events can be used to trigger a conversion. The EXTSEL[1:0] bits are used to select the two possible sources of events that can trigger conversion.

To use external trigger mode:

1. The ADC is in off state (ADON=0) and EOC bit is cleared.
2. Select trigger source (EXTSEL [1:0]).
3. Set external trigger mode EXTTRIG=1 using a BSET instruction in order not to change other bits in the register.
4. If the trigger source is in high state, this switches on the ADC. For this reason, test if ADC is switched off (ADON=0), then switch on ADC (ADON=1).
5. Wait for the stabilization time ( $t_{STAB}$ ). If an external trigger occurs before  $t_{STAB}$  elapses, the result will not be accurate.
6. Conversion starts when an external trigger event occurs.

**Note:** 1 *If timer trigger mode is selected (timer event as trigger source, not external pin) it is recommended to start the timer only when the ADC is completely set - and stop the timer before the ADC is switched off.*  
2 *External trigger mode must be disabled (EXTTRIG=0) before executing a HALT instruction.*

#### 24.5.8 Analog zooming

Analog zooming is supported in devices with external reference voltage pins ( $V_{REF+}$  and  $V_{REF-}$ ). In analog zooming, the reference voltage is chosen to allow increased resolution in a reduced voltage range. Refer to the datasheet for details on the allowed reference voltage range.

#### 24.5.9 Timing diagram

As shown in [Figure 162](#), after ADC power on, the ADC needs a stabilization time  $t_{STAB}$  (equivalent to one conversion time  $t_{CONV}$ ) before it starts converting accurately. For subsequent conversions there is no stabilization delay and ADON needs to be set only once. The ADC conversion time takes 14 clock cycles. After conversion the EOC flag is set and the 10-bit ADC Data register contains the result of the conversion.

Figure 162. Timing diagram in single mode (CONT = 0)

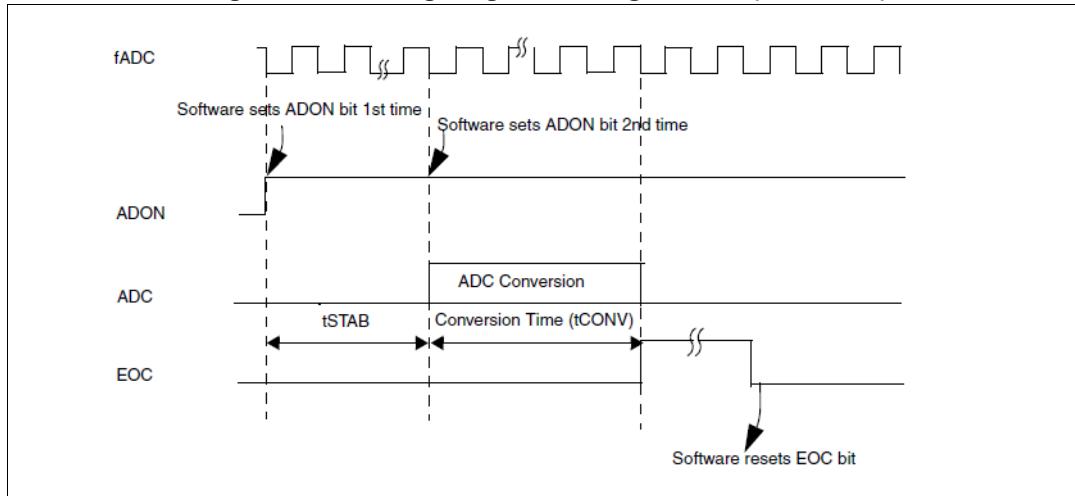
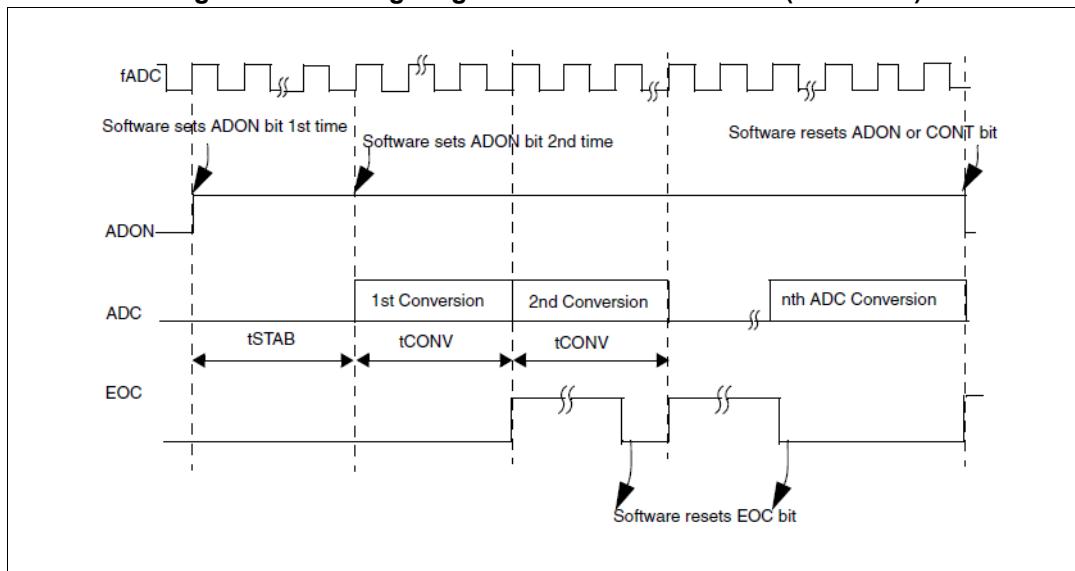


Figure 163. Timing diagram in continuous mode (CONT = 1)



## 24.6 ADC low power modes

**Table 73. Low power modes**

Mode	Description
Wait	No effect on ADC
Halt/ Active-halt	In devices with extended features, the ADC is automatically switched off before entering Halt/Active-halt mode. After waking up from Active-halt, the ADON bit must be set by software to power on the ADC, and a delay of 7 µs is needed before starting a new conversion.

The ADC does not have the capability to wake the device from Active-halt or Halt mode.

## 24.7 ADC interrupts

The ADC interrupt control bits are summarized in [Table 74](#), [Table 75](#) and [Table 76](#)

**Table 74. ADC Interrupts in single and non-buffered continuous mode (ADC1 and ADC2)<sup>(1)</sup>**

Enable bits			Status flags			Exit from Wait	Exit from Halt
AWENx	AWDIE	EOCIE	AWSx	AWDG	EOC		
Don't care	0	0	Don't care	Flag is set if the channel crosses the programmed thresholds.	Flag is set at the end of each conversion.	No	No
	0	1		Flag is set if the channel crosses the programmed thresholds.	Flag is set at the end of each conversion and an interrupt is generated.	Yes	No
	1	0		Flag is set if the channel crosses the programmed thresholds. An interrupt is generated but continuous conversion is not stopped.	Flag is set at the end of each conversion.	Yes	No
	1	1		Flag is set if the channel crosses the programmed thresholds. An interrupt is generated but continuous conversion is not stopped.	Flag is set at the end of each conversion and an interrupt is generated.	Yes	no

1. BSIZE = Data buffer size (8 or 10 depending on the product).

**Table 75. ADC interrupts in buffered continuous mode (ADC1)**

Enable bits			Status flags			Exit from Wait	Exit from Halt
<b>AWENx</b>	<b>AWDIE</b>	<b>EOCIE</b>	<b>AWSx</b>	<b>AWD</b>	<b>EOC</b>		
0	Don't care	0	0	0	The flag is set at the end of BSIZE conversions	No	No
0	Don't care	1	0		The flag is set at the end of BSIZE conversions and an interrupt is generated.	Yes	No
1	0	0	Flag is set if conversion on buffer "x" crosses the thresholds programmed in the ADC_HTR and ADC_LTR registers	The flag is set at the end of BSIZE conversions if at least one of the AWSx bits is set	The flag is set at the end of BSIZE conversions (Data Buffer Full)	No	No
1	1	0		The flag is set and an interrupt is generated at the end of BSIZE conversions if at least one of the AWSx bits is set. Continuous conversion is not stopped.		Yes	No
1	0	1		The flag is set at the end of BSIZE conversions if at least one of the AWSx bits is set	The flag is set at the end of BSIZE conversions and an interrupt is generated.	Yes	No
1	1	1		The flag is set immediately as soon as one of the AWSx bits is set. In interrupt is generated and continuous conversion is stopped.	The flag is set at the end of BSIZE conversions and an interrupt is generated.	Yes	No

**Table 76. ADC interrupts in scan mode (ADC1)**

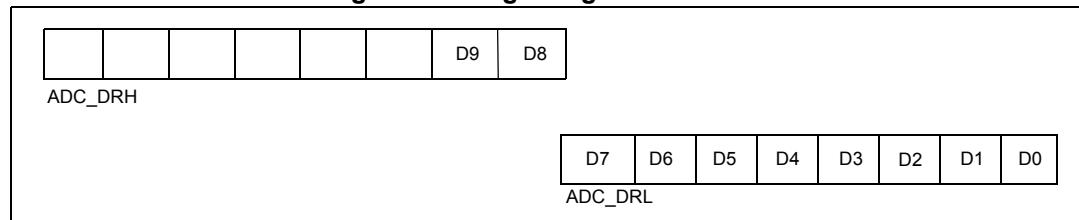
Control bits			Status bits			Exit from Wait	Exit from Halt
<b>AWENx</b>	<b>AWDIE</b>	<b>EOCIE</b>	<b>AWSx</b>	<b>AWD</b>	<b>EOC</b>		
0	Don't care	0	0	0	The flag is set at the end of the scan sequence	No	No
0	Don't care	1	0	0	The flag is set at the end of the scan sequence and an interrupt is generated.	Yes	No
1	0	0	Flag is set if conversion on channel "x" crosses the thresholds programmed in the ADC_HTR and ADC_LTR registers	The flag is set at the end of the scan sequence if at least one of the AWSx bits is set	The flag is set at the end of the scan sequence	No	No
1	1	0		The flag is set and an interrupt is generated at the end of the SCAN sequence if at least one of the AWSx bits is set. SCAN conversion is not stopped.	The flag is set to 1 at the end of the scan sequence	Yes	No
1	0	1		The flag is set at the end of the scan sequence if at least one of the AWSx bits is set	The flag is set to 1 at the end of the scan sequence and an interrupt is generated.	Yes	No
1	1	1		The flag is set immediately as soon as one of the AWSx bits is set. In interrupt is generated and scan conversion is stopped.	The flag is set at the end of the scan sequence and an interrupt is generated.	Yes	No

## 24.8 Data alignment

ALIGN bit in the ADC\_CR2 register selects the alignment of data stored after conversion. Data can be aligned in the following ways.

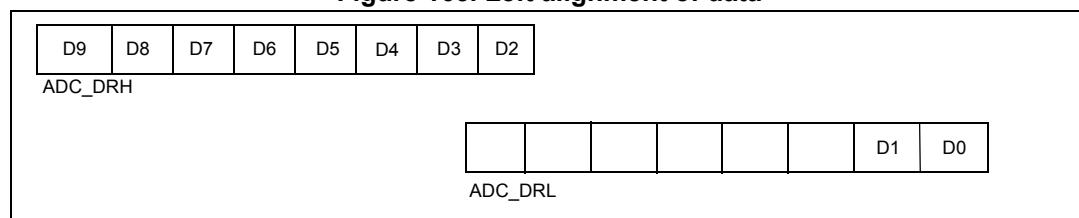
**Right Alignment:** 8 Least Significant bits are written in the ADC\_DL register, then the remaining Most Significant bits are written in the ADC\_DH register. The Least Significant Byte must be read first followed by the Most Significant Byte. In this case, the LDW instruction can be used as it has the same reading order.

Figure 164. Right alignment of data



**Left Alignment:** 8 Most Significant bits are written in the ADC\_DH register, then the remaining Least Significant bits are written in the ADC\_DL register. The Most Significant Byte must be read first followed by the Least Significant Byte.

Figure 165. Left alignment of data



## 24.9 Reading the conversion result

The conversion results from ADC\_DRH and ADC\_DRL data registers must be read in a specific order to guarantee data coherency. This order depends on the data alignment (refer to [Section 24.8: Data alignment](#)).

When the ADC1DBxRH and ADC1DBxRL data buffer registers are read (ADC1 only), there is no internal locking mechanism. Therefore, the user must check the OVR flag in the ADC\_CR3 register after having read the ADC1DBxRH and AD1CDBxRL registers. If the OVR flag is cleared, this ensures that the values just read from the ADC1DBxRH and AD1CDBxRL registers are consistent.

Another way to ensure data consistency (with right alignment of data) is to read ADC1DBxRH and ADC1DBxRL with the following sequence, which must not be interrupted:

```
ADC_READ:  
LDW X,#ADC_DB0RH  
CPW X,#ADC_DB0RH  
JREQ ADC_END  
LDW X,#ADC_DB0RH  
ADC_END:
```

## 24.10 Schmitt trigger disable registers

The ADC\_TDRH and ADC\_TDRL registers are used to disable the Schmitt triggers available in the AIN analog input pins. Disabling the Schmitt trigger lowers the power consumption in the I/Os.

## 24.11 ADC registers

### 24.11.1 ADC data buffer register x high (ADC\_DBxRH) (x=0..7 or 0..9 )

Address offset: 0x00 + 2 \* channel number

Reset value: 0x00

7	6	5	4	3	2	1	0
DBH[7:0]							
r	r	r	r	r	r	r	r

**Note:** *Data buffer registers are not available for ADC2. The data buffer size and base address are device dependent and are specified in the corresponding datasheet. Note that the data buffer registers and the other ADC registers have different base addresses.*

#### Bits 7:0 DBH[7:0] Data bits high

These bits are set/reset by hardware and are read only. When the ADC is in buffered continuous or scan mode, they contain the high part of the converted data. The data is in right-aligned or left-aligned format depending on the ALIGN bit.

#### Left Data Alignment

These bits contain the eight MSB bits of the converted data.

#### Right Data Alignment

These bits contain the (eight ADC data width) MSB bits of the converted data. Remaining bits are tied to zero.

See [Figure 164](#).

### 24.11.2 ADC data buffer register x low (ADC\_DBxRL) (x=or 0..7 or 0..9)

Address offset: 0x01 + 2 \* channel number

Reset value: 0x00

7	6	5	4	3	2	1	0
DBL[7:0]							
r	r	r	r	r	r	r	r

Note: *Data buffer registers are not available for ADC2. The data buffer size and base address are device dependent and are specified in the corresponding datasheet. Note that the data buffer registers and the other ADC registers have different base addresses.*

#### Bits 7:0 DBL[7:0] Data bits low

These bits are set/reset by hardware and are read only. When the ADC is in buffered continuous or scan mode, they contain the low part of the A/D conversion result, in right-aligned or left-aligned format depending on the ALIGN bit.

##### Left Data Alignment

These bits contain the (eight ADC data width) LSB bits of the converted data. Remaining bits of the register are tied to zero.

See [Figure 165](#).

##### Right Data Alignment

These bits contain the eight LSB bits of the converted data.

### 24.11.3 ADC control/status register (ADC\_CSR)

Address offset: 0x20

Reset value: 0x00

7	6	5	4	3	2	1	0			
EOC	AWD	EOCIE	AWDIE	CH[3:0]						
rw	rc_w0	rw	rw	rw	rw	rw	rw			

Bit 7 **EOC**: End of conversion

This bit is set by hardware at the end of conversion. It is cleared by software by writing '0'.

0: Conversion is not complete

1: Conversion complete

Bit 6 **AWD**: Analog Watchdog flag

0: No analog watchdog event

1: An analog watchdog event occurred. In buffered continuous or scan mode you can read the ADC\_AWSR register to determine the data buffer register related to the event. An interrupt request is generated if AWDIE=1.

*Note: This bit is not available for ADC2*

Bit 5 **EOCIE**: Interrupt enable for EOC

This bit is set and cleared by software. It enables the interrupt for End of Conversion.

0: EOC interrupt disabled

1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

Bit 4 **AWDIE**: Analog watchdog interrupt enable

0: AWD interrupt disabled.

1: AWD interrupt enabled

*Note: This bit is not available for ADC2*

Bits 3:0 **CH[3:0]**: Channel selection bits

These bits are set and cleared by software. They select the input channel to be converted.

0000: Channel AIN0

0001: Channel AIN1

....

1111: Channel AIN15

#### 24.11.4 ADC configuration register 1 (ADC\_CR1)

Address offset: 0x21

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	SPSEL[2:0]					Reserved	CONT
r	rw	rw	rw		r	rw	rw

Bit 7 Reserved, always read as 0.

##### Bits 6:4 SPSEL[2:0]: Prescaler selection

These control bits are written by software to select the prescaler division factor.

- 000:  $f_{ADC} = f_{MASTER}/2$
- 001:  $f_{ADC} = f_{MASTER}/3$
- 010:  $f_{ADC} = f_{MASTER}/4$
- 011:  $f_{ADC} = f_{MASTER}/6$
- 100:  $f_{ADC} = f_{MASTER}/8$
- 101:  $f_{ADC} = f_{MASTER}/10$
- 110:  $f_{ADC} = f_{MASTER}/12$
- 111:  $f_{ADC} = f_{MASTER}/18$

See [Section 24.5.2 on page 431](#).

*Note:* It is recommended to change the SPSEL bits when ADC is in power down. This is because internally there can be a glitch in the clock during this change. Otherwise the user is required to ignore the 1st converted result if the change is done when ADC is not in power down.

Bits 3:2 Reserved, always read as 0.

##### Bit 1 CONT: Continuous conversion

This bit is set and cleared by software. If set, conversion takes place continuously till this bit is reset by software.

- 0: Single conversion mode
- 1: Continuous conversion mode

##### Bit 0 ADON: A/D Converter on/off

This bit is set and reset by software. This bit must be written to wake up the ADC from power down mode and to trigger the start of conversion. If this bit holds a value of 0 and a 1 is written to it then it wakes the ADC from power down mode. Conversion starts when this bit holds a value of 1 and a 1 is written to it. As soon as the ADC is powered on, the output stage of the selected channel is disabled.

0: Disable ADC conversion/calibration and go to power down mode.

1: Enable ADC and to start conversion

*Note:* If any other bit in this register apart from ADON is changed at the same time, then conversion is not triggered. This is to prevent triggering an erroneous conversion.

### 24.11.5 ADC configuration register 2 (ADC\_CR2)

Address offset: 0x22

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	EXTTRIG	EXTSEL[1:0]		ALIGN	Reserved	SCAN	Reserved
r	rw	rw	rw	rw	r	rw	r

Bit 7 Reserved, must be kept cleared.

Bit 6 **EXTTRIG**: External trigger enable

This bit is set and cleared by software. It is used to enable an external trigger to trigger a conversion.

0: Conversion on external event disabled

1: Conversion on external event enabled

*Note:* To avoid a spurious trigger event, use the BSET instruction to set EXTTRIG without changing other bits in the register.

Bits 5:4 **EXTSEL[1:0]**: External event selection

The two bits are written by software. They select one of four types of event used to trigger the start of ADC conversion.

00: Internal TIM1 TRGO event

01: External interrupt on ADC\_ETR pin

10: Reserved

11: Reserved

Bit 3 **ALIGN**: Data alignment

This bit is set and cleared by software.

0: Left alignment (the eight MSB bits are written in the ADC\_DRH register then the remaining LSB bits are written in the ADC\_DRL register). The reading order should be MSB first and then LSB.

1: Right alignment (eight LSB bits are written in the ADC\_DRL register then the remaining MSB bits are written in the ADC\_DH register). The reading order should be LSB first and then MSB.

*Note:* The ALIGN bit influences the ADC\_DRH/ADC\_DRL register reading order and not the reading order of the buffer registers.

Bit 2 Reserved, must be kept cleared.

Bit 1 **SCAN**: Scan mode enable

This bit is set and cleared by software.

0: Scan mode disabled

1: Scan mode enabled

*Note:* This bit is not available for ADC2

Bit 0 Reserved, must be kept cleared.

### 24.11.6 ADC configuration register 3 (ADC\_CR3)

Address offset: 0x23

Reset value: 0x00

7	6	5	4	3	2	1	0
DBUF	OVR			Reserved			
rw	rc_w0			r			

**Note:** *This register is not available for ADC2.*

**Bit 7 DBUF:** Data buffer enable

This bit is set and cleared by software. It is used together with the CONT bit enable buffered continuous mode (DBUF=1, CONT=1). When DBUF is set, converted values are stored in the ADC\_DBxRH and ADC\_DBxRL registers instead of the ADC\_DRH and ADC\_DRL registers.

- 0: Data buffer disabled
- 1: Data buffer enabled

**Bit 6 OVR:** Overrun flag

This bit is set by hardware and cleared by software.

- 0: No overrun
- 1: An overrun was detected in the data buffer registers.  
Refer to [Section 24.5.5 on page 433](#) for more details.

Bits 5:0 Reserved, must be kept cleared.

### 24.11.7 ADC data register high (ADC\_DRH)

Address offset: 0x24

Reset value: 0XX

7	6	5	4	3	2	1	0
DH[7:0]							
r	r	r	r	r	r	r	r

Bits 7:0 **DH[7:0]** Data bits high

These bits are set/reset by hardware and are read only. When the ADC is in single or non-buffered continuous mode, they contain the high part of the converted data, in right-aligned or left-aligned format depending on the ALIGN bit.

- **Left Data Alignment**

These bits contain the 8 MSB bits of the converted data. The MSB must be read first before reading the LSB (see [Section 24.9: Reading the conversion result](#) and [Figure 165](#).)

- **Right Data Alignment**

These bits contain the (ADC data width - 8) MSB bits of the converted data. Remaining bits are tied to zero.

See [Figure 164](#).

### 24.11.8 ADC data register low (ADC\_DRL)

Address offset: 0x25

Reset value: 0XX

7	6	5	4	3	2	1	0
DL[7:0]							
r	r	r	r	r	r	r	r

Bits 7:0 **DL[7:0]** Data bits low

These bits are set/reset by hardware and are read only. When the ADC is in single or non-buffered continuous mode, they contain the low part of the A/D conversion result, in right-aligned or left-aligned format depending on the ALIGN bit.

- **Left Data Alignment**

These bits contain the (ADC data width - 8) LSB bits of the converted data, remaining bits of the register are tied to zero.

See [Figure 165](#).

- **Right Data Alignment**

These bits contain the 8 LSB bits of the converted data. The LSB must be read first before reading the MSB (see [Section 24.9: Reading the conversion result](#) and [Figure 164](#).)

### 24.11.9 ADC Schmitt trigger disable register high (ADC\_TDRH)

Address offset: 0x26

Reset value: 0x00

7	6	5	4	3	2	1	0
TD[15:8]							
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **TD[15:8]** Schmitt trigger disable high

These bits are set and cleared by software. When a TDx bit is set, it disables the I/O port input Schmitt trigger of the corresponding ADC input channel x even if this channel is not being converted. This is needed to lower the static power consumption of the I/O port.

- 0: Schmitt trigger enabled
- 1: Schmitt trigger disabled

### 24.11.10 ADC Schmitt trigger disable register low (ADC\_TDRL)

Address offset: 0x27

Reset value: 0x00

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

Bits 7:0 **TD[7:0]** Schmitt trigger disable low

These bits are set and cleared by software. When a TDx bit is set, it disables the I/O port input Schmitt trigger of the corresponding ADC input channel x even if this channel is not being converted. This is needed to lower the static power consumption of the I/O port.

- 0: Schmitt trigger enabled
- 1: Schmitt trigger disabled

### 24.11.11 ADC high threshold register high (ADC\_HTRH)

Address offset: 0x28

Reset value: 0xFF

7	6	5	4	3	2	1	0
HT[9:2]							
rw	rw	rw	rw	rw	rw	rw	rw

*Note:* This register is not available for ADC2.

Bits 7:0 **HT[9:2]** Analog Watchdog High Voltage threshold MSB

These bits are set and cleared by software. They define the MSB of the high threshold ( $V_{REFH}$ ) for the Analog Watchdog.

### 24.11.12 ADC high threshold register low (ADC\_HTRL)

Address offset: 0x29

Reset value: 0x03

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

*Note:* This register is not available for ADC2.

Bits 7:2 Reserved, must be kept cleared.

Bits 1:0 **HT[1:0]** Analog watchdog high voltage threshold LSB

These bits are set and cleared by software. They define the LSB of the high threshold ( $V_{REFH}$ ) for the Analog Watchdog.

### 24.11.13 ADC low threshold register high (ADC\_LTRH)

Address offset: 0x2A

Reset value: 0x00

7	6	5	4	3	2	1	0
LT[9:2]							
rw	rw	rw	rw	rw	rw	rw	rw

**Note:** *This register is not available for ADC2.*

Bits 7:0 **LT[9:2]** Analog watchdog low voltage threshold MSB

These bits are set and cleared by software. They define the MSB of the low Threshold ( $V_{REFL}$ ) for the Analog Watchdog.

### 24.11.14 ADC low threshold register low (ADC\_LTRL)

Address offset: 0x2B

Reset value: 0x00

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

**Note:** *This register is not available for ADC2.*

Bits 7:2 Reserved, must be kept cleared.

Bits 1:0 **LT[1:0]** Analog watchdog low voltage threshold LSB

These bits are set and cleared by software. They define the LSB of the low threshold ( $V_{REFL}$ ) for the Analog Watchdog.

### 24.11.15 ADC watchdog status register high (ADC\_AWSRH)

Address offset: 0x2C

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						AWS[9:8]	
r						rc_w0	rc_w0

**Note:** *This register is not available for ADC2.*

Bits 7:2 Reserved, must be kept cleared.

Bits 1:0 **AWS[9:8]** Analog watchdog status flags 9:8

These bits are set by hardware and cleared by software.

- In buffered continuous mode (DBUF=1, CONT=1) AWS flags behave as described in [Table 75](#).
- In scan mode (SCAN=1) AWS flags behave as described in [Table 76](#).

0: No analog watchdog event in data buffer register x.

1: Analog watchdog event occurred in data buffer register x.

### 24.11.16 ADC watchdog status register low (ADC\_AWSRL)

Address offset: 0x2D

Reset value: 0x00

7	6	5	4	3	2	1	0
AWS[7:0]							
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

**Note:** *This register is not available for ADC2.*

Bits 7:0 **AWS[7:0]** Analog watchdog status flags 7:0

These bits are set by hardware and cleared by software.

- In buffered continuous mode (DBUF=1, CONT=1) AWS flags behave as described in [Table 75](#).
- In scan mode (SCAN=1) AWS flags behave as described in [Table 76](#).

0: No analog watchdog event in data buffer register x.

1: Analog watchdog event occurred in data buffer register x.

### 24.11.17 ADC watchdog control register high (ADC\_AWCRH)

Address offset: 0x2E

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved						AWEN[9:8]	
r						rw	rw

Note: *This register is not available for ADC2.*

Bits 7:2 Reserved, must be kept cleared.

Bits 1:0 **AWEN[9:8]** Analog watchdog enable bits 9:8

These bits are set and cleared by software.

In buffered continuous mode (DBUF=1, CONT=1) and in scan mode (SCAN=1) the AWENx bits enable the analog watchdog function for each of the 10 data buffer registers.

0: Analog watchdog disabled in data buffer register x.

1: Analog watchdog enabled in data buffer register x.

### 24.11.18 ADC watchdog control register low (ADC\_AWCRL)

Address offset: 0x2F

Reset value: 0x00

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

Note: *This register is not available for ADC2.*

Bits 7:0 **AWEN[7:0]** Analog watchdog enable bits 7:0

These bits are set and cleared by software.

In buffered continuous mode (DBUF=1, CONT=1) and in scan mode (SCAN=1) the AWENx bits enable the analog watchdog function for each of the 10 data buffer registers.

0: Analog watchdog disabled in data buffer register x.

1: Analog watchdog enabled in data buffer register x.

## 24.12 ADC register map and reset values

Table 77. ADC1 register map and reset values

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	ADC1_DB0RH Reset value	-0	-0	-0	-0	-0	-0	DATA90	DATA80
0x01	ADC1_DB0RL Reset value	DATA70	DATA60	DATA50	DATA40	DATA30	DATA20	DATA10	DATA00
0x02 to 0x0D	Reserved								
0x0E	ADC1_DB7RH Reset value	-0	-0	-0	-0	-0	-0	DATA90	DATA80
0x0Fh	ADC1_DB7RL Reset value	DATA70	DATA60	DATA50	DATA40	DATA30	DATA20	DATA10	DATA00
0x10	ADC1_DB8RH <sup>(1)</sup> Reset value	-0	-0	-0	-0	-0	-0	DATA90	DATA80
0x11	ADC1_DB8RL <sup>(1)</sup> Reset value	DATA70	DATA60	DATA50	DATA40	DATA30	DATA20	DATA10	DATA00
0x12	ADC1_DB9RH <sup>(1)</sup> Reset value	-0	-0	-0	-0	-0	-0	DATA90	DATA80
0x13	ADC1_DB9RL <sup>(1)</sup> Reset value	DATA70	DATA60	DATA50	DATA40	DATA30	DATA20	DATA10	DATA00
0x14 to 0x1F	Reserved								
0x20	ADC1_CSR Reset value	EOC0	AWD0	EOCIE0	AWDIE0	CH30	CH20	CH10	CH00
0x21	ADC1_CR1 Reset value	-0	SPSEL20	SPSEL10	SPSEL00	-0	-0	CONT0	ADON0
0x22	ADC1_CR2 Reset value	-0	EXTTRIG0	EXTSEL10	EXTSEL00	ALIGN0	-0	SCAN0	-0
0x23	ADC1_CR3 Reset value	DBUF0	OVR0	-0	-0	-0	-0	-0	-0
0x24	ADC1_DRH Reset value	-x	-x	-x	-x	-x	-x	DATA9x	DATA8x
0x25	ADC1_DRL Reset value	DATA7x	DATA6x	DATA5x	DATA4x	DATA3x	DATA2x	DATA1x	DATA0x
0x26	ADC1_TDRH <sup>(2)</sup> Reset value	TD150	TD140	TD130	TD120	TD110	TD100	TD90	TD80
0x27	ADC1_TDRL Reset value	TD70	TD60	TD50	TD40	TD30	TD20	TD10	TD00
0x28	ADC1_HTRH Reset value	HT91	HT81	HT71	HT61	HT51	HT41	HT31	HT21
0x29	ADC1_HTRL Reset value	-0	-0	-0	-0	-0	-0	HT11	HT01
0x2A	ADC1_LTRH Reset value	LT90	LT80	LT70	LT60	LT50	LT40	LT30	LT20
0x2B	ADC1_LTRL Reset value	-0	-0	-0	-0	-0	-0	LT10	LT00
0x2C	ADC1_AWSRH <sup>(2)</sup> Reset value	-0	-0	-0	-0	-0	-0	AWS90	AWS80

**Table 77. ADC1 register map and reset values (continued)**

<b>Address offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x2D	ADC1_AWSRL Reset value	AWS7 0	AWS6 0	AWS5 0	AWS4 0	AWS3 0	AWS2 0	AWS1 0	AWS0 0
0x2E	ADC1_AWCRH <sup>(2)</sup> Reset value	- 0	- 0	- 0	- 0	- 0	- 0	AWEN9 0	AWEN8 0
0x2F	ADC1_AWCRL Reset value	AWEN7 0	AWEN6 0	AWEN5 0	AWEN4 0	AWEN3 0	AWEN2 0	AWEN1 0	AWEN0 0

1. This register is reserved in devices with buffer size 8 x 10 bits.

2. This register is reserved in devices without ADC channels 8 and 9.

**Table 78. ADC2 register map and reset values**

<b>Address offset</b>	<b>Register name</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
0x20	ADC2_CSR Reset value	EOC 0	AWD 0	EOCIE 0	AWDIE 0	CH3 0	CH2 0	CH1 0	CH0 0
0x21	ADC2_CR1 Reset value	- 0	SPSEL2 0	SPSEL1 0	SPSEL0 0	- 0	- 0	CONT 0	ADON 0
0x22	ADC2_CR2 Reset value	- 0	EXTTRIG 0	EXTSEL1 0	EXTSEL0 0	ALIGN 0	- 0	- 0	- 0
0x23	ADC2_CR3 Reset value	DBUF 0	OVR 0	- 0	- 0	- 0	- 0	- 0	- 0
0x24	ADC2_DRH Reset value	- 0	- 0	- 0	- 0	- 0	- 0	DATA9 0	DATA8 0
0x25	ADC2_DRL Reset value	DATA7 0	DATA6 0	DATA5 0	DATA4 0	DATA3 0	DATA2 0	DATA1 0	DATA0 0
0x26	ADC2_TDRH Reset value	TD15 0	TD14 0	TD13 0	TD12 0	TD11 0	TD10 0	TD9 0	TD8 0
0x27	ADC2_TDRL Reset value	TD7 0	TD6 0	TD5 0	TD4 0	TD3 0	TD2 0	TD1 0	TD0 0

## 25 Revision history

**Table 79. Document revision history**

Date	Revision	Changes
27-May-2008	1	Initial release.
13-Aug-2008	2	<p>Updated <a href="#">Section 2: Memory and register map on page 27</a>: introduced high, medium and low density categories; modified end address for option bytes; updated RAM, data EEPROM and Flash program memory densities.</p> <p>Updated <a href="#">Figure 18: Reset circuit on page 74</a></p> <p>Update min reset pulse from 300 to 500 ns in <a href="#">Section 8.2: Reset circuit description on page 74</a></p> <p>Updated <a href="#">Table 6: Memory access versus programming method on page 50</a>.</p> <p>Reorganized <a href="#">Section 16 on page 134</a> to <a href="#">Section 19 on page 253</a></p> <p>Renamed USART and LINUART to UART1, UART2 and UART3 combined in new <a href="#">Section 22 on page 320</a>.</p> <p>Updated CAN filter and external clock description in <a href="#">Section 23 on page 378</a>.</p> <p>Renamed ADC to ADC1 and ADC2 in <a href="#">Section 24 on page 428</a></p> <p>Updated <a href="#">Continuous scan mode on page 433</a></p> <p>Updated <a href="#">Conversion on external trigger on page 435</a></p>
22-Sep-2008	3	<p>Updated <a href="#">Section 4: Flash program memory and data EEPROM</a>.</p> <p>Changed name of SWUAH bit to REGAH in <a href="#">Section 9.9.1: Internal clock register (CLK_ICKR) on page 90</a>.</p> <p>Modified LSI frequency measurement in <a href="#">Section 11.1 on page 105</a></p> <p>Modified <a href="#">Peripheral clock gating register 1 (CLK_PCKENR1) on page 95</a></p> <p>Modified <a href="#">Section 11.8.2: Slope control on page 110</a>.</p> <p>Added description of TIM5, TIM6 in <a href="#">Section 16: Timer overview</a>, <a href="#">Section 18: 16-bit general purpose timers (TIM2, TIM3, TIM5)</a> and <a href="#">Section 19: 8-bit basic timer (TIM4, TIM6)</a>.</p> <p>Updated <a href="#">Section 24.5.6: Analog watchdog</a>.</p>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
15-Jan-2009	4	<p>Removed memory and register map (information transferred to datasheets)</p> <p>Register absolute addresses replaced by offsets. (refer now to register map in datasheet for the base addresses).</p> <p>Added <a href="#">Note</a> related to TLI interrupt in <a href="#">Section 6.2.1 on page 60</a>.</p> <p>Added TLI in <a href="#">Section 6.5: Concurrent and nested interrupt management</a>.</p> <p>Updated Flash program density to 32 - 128 Kbytes for high density STM8S devices in <a href="#">Section 4: Flash program memory and data EEPROM</a>.</p> <p>Updated size of STM8S option byte area in <a href="#">Section 4.4: Memory organization</a> and <a href="#">Figure 6</a>, <a href="#">Figure 7</a>, and <a href="#">Figure 8</a>.</p> <p>Updated maximum value of UBC in <a href="#">Figure 11</a>. Added information on DATA area programming on devices with and without RWW capability in <a href="#">Section 4.6.2: Byte programming</a> and <a href="#">Section 4.6.4: Block programming</a>.</p> <p>Added HVOFF in: <a href="#">Fast block programming</a>, <a href="#">Fast block programming</a>, and <a href="#">Section 4.8.8: Flash status register (FLASH_IAPSR)</a>. Updated bitfield access types in <a href="#">Section 4.8.8: Flash status register (FLASH_IAPSR) on page 55</a>.</p> <p><a href="#">Table 6: Memory access versus programming method</a>: removed NMI and TRAP vectors, modified access for option bytes in ICP/SWIM mode/ROP enabled, and UBC ROP disabled.</p> <p>Updated <a href="#">Table 28: Watchdog timeout period (LSI clock frequency = 128 kHz) on page 125</a></p> <p>Updated <a href="#">Table 29: Approximate timeout duration on page 130</a></p> <p><a href="#">Table 30: Window watchdog timing diagram on page 131</a></p> <p>Updated <a href="#">Note 8 on page 311</a></p>
10-Aug-2009	5	<p>Added note to <a href="#">Section 4.4: Memory organization</a>.</p> <p>Added <a href="#">Section 4.4.2: Memory access/ wait state configuration</a>.</p> <p>Updated maximum value of UCB[7:0] in <a href="#">Figure 2: Page 255 is reserved for data EEPROM</a>.</p> <p>Added note 1 below <a href="#">Figure 10</a>. Added note 1 and updated note 3 below <a href="#">Figure 11</a>.</p> <p>Check in PUL/DUL bits made mandatory in <a href="#">Section 4.5.2: Memory access security system (MASS)</a>.</p> <p>Added details in <a href="#">Section 4.6: Memory programming</a> on word programming in main program and DATA.</p> <p>Updated <a href="#">Section 4.8.8: Flash status register (FLASH_IAPSR) on page 55</a>.</p> <p>Added note to <a href="#">Section 9.1.2: HSI (high-speed internal) clock signal</a>.</p> <p>Updated <a href="#">Table 17</a> (UART peripheral clock gating bit description moved to datasheet).</p> <p>Updated <a href="#">Table 20: Low power mode management on page 102</a></p> <p>Updated management of hardware interrupts in <a href="#">Section 6.1: ITC introduction</a>.</p> <p>Removed interrupt vector table (moved to datasheet)</p>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
10-Aug-2009	5 cont'd	<p>Changed note in <a href="#">Section 6.9.2: Software priority register x (ITC_SPRx) on page 69</a>.</p> <p>Updated AWU <a href="#">Section 12.3.2: Time base selection</a>.</p> <p>Removed description of timer input XOR feature (TI1S bit in <a href="#">Section 17</a> and <a href="#">Section 18</a>).</p> <p>Updated trigger selection for and ETR description for TIM5 in <a href="#">Section 18</a>.</p> <p>Updated MMS bits in <a href="#">Control register 2 (TIM5_CR2)</a> and <a href="#">Control register 2 (TIM6_CR2)</a>.</p> <p>Updated TG bit of <a href="#">Event generation register (TIMx_EGR)</a>.</p> <p>Added note on TIM2 and TIM4 register offsets in <a href="#">Section on page 247</a> and <a href="#">Section 19.6.10 on page 260</a></p> <p><a href="#">Section 21.4.3, Acknowledge failure (AF)</a>: Added “repeated start” to master condition.</p> <p>Modified <a href="#">Section 21.7.3: Frequency register (I2C_FREQR) on page 307</a>.</p> <p>Added 6th step to UART <a href="#">Character transmission</a>. Updated UART <a href="#">Single byte communication</a>.</p> <p>Added <a href="#">Figure 116: TC/TXE behavior when transmitting</a>.</p> <p>Updated TC bit description in <a href="#">Section 22.7.1: Status register (UART_SR)</a>.</p> <p>Added <a href="#">Start bit detection</a> and <a href="#">Section 22.3.5: Clock deviation tolerance of the UART receiver</a> in <a href="#">Section 22.3.3: Receiver</a>.</p> <p>Added a caution to <a href="#">Section 23.11.15: Mailbox registers</a>.</p> <p>Updated description of TGT in <a href="#">CAN mailbox data length control register (CAN_MDLCR)</a>.</p> <p>Changed alignment of threshold registers and added note for data buffer base address in <a href="#">Section 24.11: ADC registers</a>.</p>
08-Dec-2009	6	<p><a href="#">Peripheral clock gating register 2 (CLK_PCKENR2)</a>: Replaced address offset.</p> <p><a href="#">Table 20: Low power mode management</a>: Updated peripheral information for Active halt (--), Active halt with MVR auto power off (- -), and Halt (----).</p> <p><a href="#">Repetition counter register (TIM1_RCR)</a>: Replaced the reset value.</p> <p><a href="#">Interrupt enable register (TIMx_IER)</a>: Corrected name of bit 3 (CC3IE) in register table.</p> <p><a href="#">Status register 1 (TIMx_SR1)</a>: Added description of bit 3 (CC3IF) to register description table.</p> <p><a href="#">Figure 93: Data clock timing diagram</a>: Removed “from master” and “from slave” beneath MISO and MOSI respectively.</p> <p><a href="#">Section 20.3.5: Data transmission and reception procedures</a>: timing diagrams revised and description of receive-only mode expanded.</p> <p>Added <a href="#">Section 20.3.8: Disabling the SPI</a></p> <p><a href="#">Master mode fault (MODF)</a>: SPE and MSTR bits can be returned to their original state only after a MODF bit clearing sequence.</p> <p><a href="#">SPI interrupt control register (SPI_ICR)</a>: Removed notes relating to the TXIE and RXIE bits.</p>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
08-Dec-2009	6 cont'd	<p><i>Figure 105: Method 1: transfer sequence diagram for master receiver:</i> Added footnote concerning the next data reception and the EV7event.</p> <p><i>Bus error (BERR):</i> Updated.</p> <p>Updated <i>Figure 116: TC/TXE behavior when transmitting</i> and removed note concerning IDLE preamble.</p> <p>Updated <i>Section 24.9: Reading the conversion result</i> to account for the fact that the reading order of the ADC results from the buffer registers has no impact on data coherency.</p> <p><i>Section 24.11.1</i> and <i>Section 24.11.2:</i> Removed sentence about the reading order of the MSB and LSB bits respectively.</p> <p><i>Section 24.11.5:</i> Added note about the ALIGN bit reading order.</p>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
31-Jan-2011	7	<p>Merge with STM8A reference manual (RM0009).          Renamed low power modes, Halt, Active-halt, Wait, and Run in the whole document.          Added overview of STM8S and STM8A device families on coverpage.</p> <p><b>Section 2: Boot ROM:</b> added LIN mode configuration.</p> <p><b>Section 3: Memory and register map:</b></p> <ul style="list-style-type: none"> <li>– Updated <a href="#">Section 3.1.1: Memory map</a> to cover both STM8A and STM8S devices.</li> <li>– Added <a href="#">Section 3.1.2: Stack handling</a>.</li> </ul> <p><b>Section 4: Flash program memory and data EEPROM</b></p> <ul style="list-style-type: none"> <li>– Updated Flash program memory and SRAM size for medium density STM8S and STM8A devices in <a href="#">Section 4.4.1: STM8S and STM8AF memory organization</a>.</li> <li>– Added <a href="#">Note 1</a> below <a href="#">Figure 7: Flash memory and data EEPROM organization on medium density STM8S and STM8AF</a>.</li> <li>– Added 32 Kbyte and 128 Kbyte STM8A devices and <a href="#">Section 4.4.2: Memory access/ wait state configuration</a>.</li> <li>– <a href="#">Section 4.5.1: Readout protection</a>: added DM mode when readout protection is enabled, and <a href="#">Section : Temporarily removing the readout protection</a>.</li> <li>– Added case of FLASH_CR1/FLASH_CR2 access during memory write operation in <a href="#">Section 4.6.1: Read-while-write (RWW)</a>.</li> <li>– Moved all information related to temporary memory unprotection to dedicated application note.</li> </ul> <p><b>Section 7: Power supply:</b></p> <ul style="list-style-type: none"> <li>– Removed minimum VCAP value in Power section overview.</li> </ul> <p><b>Section 8: Reset (RST):</b></p> <ul style="list-style-type: none"> <li>– Changed EMS to EMC.</li> <li>– Added <a href="#">Section 8.1: "Reset state" and "under reset" definitions</a>.</li> <li>– Replace numerical values by <math>t_{OP(NRST)}</math> and <math>t_{INFP(NRST)}</math> in <a href="#">Section 8.2: Reset circuit description</a>.</li> </ul> <p><b>Section 9: Clock control (CLK):</b></p> <ul style="list-style-type: none"> <li>– Added <a href="#">Table 14: Devices with 4 trimming bits</a> and <a href="#">Table 15: Devices with 3 trimming bits</a> in <a href="#">Section 9.1.2: HSI (high-speed internal) clock signal</a>.</li> <li>– Updated CLK_HSITRIMR, CLK_SWIMCCR, and CLK_SWCR reset values.</li> </ul> <p><b>Section 10: Power management:</b></p> <ul style="list-style-type: none"> <li>– Updated Fast clock wakeup in <a href="#">Section 10.2.3: Active-halt modes</a>.</li> </ul> <p><b>Section 6: Interrupt controller (ITC):</b></p> <ul style="list-style-type: none"> <li>– Added caution note concerning interrupt disabling inside an ISR in <a href="#">Section 6.2: Interrupt masking and processing flow</a>.</li> <li>– Added Push CC instruction in <a href="#">Table 11: Dedicated interrupt instruction set</a>.</li> <li>– Removed note 3 in <a href="#">Section 6.2.1: Servicing pending interrupts</a>.          Removed case of TRAP interruption by TLI in <a href="#">Section 6.2.2: Interrupt sources</a>.</li> </ul>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
31-Jan-2011	7 cont'd	<ul style="list-style-type: none"> <li>– Removed Halt mode and HALT instruction from <a href="#">Section 6.4: Activation level/low power mode control</a>.</li> <li><b>Section 11: General purpose I/O ports (GPIO):</b> <ul style="list-style-type: none"> <li>– Added note <a href="#">Figure 24: GPIO block diagram</a>.</li> <li>– Removed warning note in <a href="#">Section 11.3: Port configuration and usage</a>.</li> <li>– Updated <a href="#">Table 21: I/O port configuration summary</a>.</li> <li>– Updated <a href="#">Section 11.4: Reset configuration</a>.</li> <li>– Updated unused I/O pin status in <a href="#">Section 11.5: Unused I/O pins</a>.</li> <li>– Added TLI masking in <a href="#">Section 11.7.2: Interrupt capability</a>.</li> <li>– Updated <a href="#">Section 11.7.3: Analog channels</a>.</li> <li>– Updated <a href="#">Section 11.8.2: Slope control</a>.</li> <li>– Changed reset value of Px_IDR to 0xXX in <a href="#">Section 11.9.2: Port x pin input register (Px_IDR)</a>.</li> <li>– Specified PD_CR1 reset value in <a href="#">Section 11.9.4: Port x control register 1 (Px_CR1)</a>.</li> </ul> </li> <li><b>Section 12: Auto-wakeup (AWU):</b> <ul style="list-style-type: none"> <li>– Modified Step 5 in <a href="#">Section 12.3.1: AWU operation</a>.</li> </ul> </li> <li><b>Section 17: 16-bit advanced control timer (TIM1):</b> <ul style="list-style-type: none"> <li>– Modified <a href="#">Figure 71: Center-aligned PWM waveforms (ARR = 8)</a>.</li> <li>– Changed f<sub>SYSCLK</sub> to f<sub>MASTER</sub> in <a href="#">Figure 31: TIM1 general block diagram</a>.</li> <li>– TIM1_TRIG renamed TIM1_ETR.</li> </ul> </li> <li><b>Section 20: Serial peripheral interface (SPI):</b> <ul style="list-style-type: none"> <li>– Added note related to parallel multislave structures in <a href="#">Section 20.3.2: Configuring the SPI in slave mode</a>.</li> </ul> </li> <li><b>Section 21: Inter-integrated circuit (I2C) interface:</b> <ul style="list-style-type: none"> <li>– Modified <a href="#">Figure 101: I2C block diagram on page 290</a>, <a href="#">Figure 102: Transfer sequence diagram for slave transmitter</a> and <a href="#">Figure 103: Transfer sequence diagram for slave receiver</a>.</li> <li>– Modified <a href="#">Section 21.4.2: I2C master mode</a>.</li> <li>– Modified PO bit description changed in <a href="#">Section 21.7.2: Control register 2 (I2C_CR2)</a>.</li> <li>– Modified note 8 in <a href="#">Section 21.7.7: Status register 1 (I2C_SR1)</a>, <a href="#">Section 21.7.11: Clock control register low (I2C_CCRL)</a> and <a href="#">Section 21.7.12: Clock control register high (I2C_CCRH)</a>.</li> <li>– Added <a href="#">Table 50: I2C_CCR values for SCL frequency table (f<sub>MASTER</sub> = 10 MHz or 16 MHz)</a>.</li> </ul> </li> <li><b>Section 22: Universal asynchronous receiver transmitter (UART):</b> <ul style="list-style-type: none"> <li>– Updated <a href="#">LIN break and delimiter detection</a>.</li> <li>– Updated <a href="#">Table 54: Baud rate programming and error calculation</a>.</li> <li>– Updated interrupt source flags and slave mode features updated in <a href="#">Section 22.2</a>.</li> </ul> </li> </ul>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
31-Jan-2011	7(continued)	<p><b>Section 23: Controller area network (beCAN):</b></p> <ul style="list-style-type: none"> <li>– Modified <math>f_{CANEXT}</math> upper limit in <a href="#">Section 23.9: Clock system</a>.</li> <li>– SLEEP and AWUM bit description updated in <a href="#">Section 23.11.1: CAN master control register (CAN_MCR)</a></li> <li>– External beCAN clock source (<math>f_{CANEXT}</math>) removed together with bit CLKS of CAN_BTR2 register.</li> <li>– Removed CAN register CLK_CANCCR.</li> </ul> <p><b>Section 24: Analog/digital converter (ADC):</b></p> <ul style="list-style-type: none"> <li>– Updated address offset for ADC_CSR to ADC_AWCRL.</li> <li>– Added AIN12 pin and <a href="#">Note 1</a> in <a href="#">Figure 159: ADC1 block diagram</a>, and note related to AIN12 in <a href="#">Section 24.5.4: Conversion modes</a>.</li> </ul>
15-Dec-2011	8	<p>Added value line STM8S devices on page 1</p> <p>Modified <a href="#">Section 4.4.1: STM8S and STM8AF memory organization on page 36</a></p> <p>Modified <a href="#">Section 6.6: External interrupts on page 66</a></p> <p>Modified TLIS bit description in <a href="#">Section 6.9.4: External interrupt control register 1 (EXTI_CR2) on page 71</a></p> <p>Modified <a href="#">Figure 20: Clock tree on page 79</a>.</p> <p>Modified HSE oscillator in quartz crystal configuration in <a href="#">Section 9.6: Clock security system (CSS) on page 88</a></p> <p>Removed one sentence in <a href="#">Section 11.8.1: Alternate function output on page 110</a>.</p> <p>Modified <a href="#">Timeout period on page 125</a></p> <p>Modified <a href="#">Figure 101: I2C block diagram on page 290</a> (SMBA pin removed)</p> <p>Replaced SYSCLK with <math>f_{CPU}</math> in <a href="#">Section 15: Window watchdog (WWDG) on page 128</a>.</p> <p>Modified <a href="#">Section 15.7: Using Halt mode with the WWDG (WWDGHALT option) on page 132</a></p> <p>Removed note 1 below <a href="#">Figure 101: I2C block diagram on page 290</a></p> <p>Added one note in <a href="#">Section : Output stage</a></p> <p>Added one note to OPM bit description in <a href="#">Section 18.6.1: Control register 1 (TIMx_CR1)</a></p> <p>Note added below <a href="#">Section 21.7.9: Status register 3 (I2C_SR3) on page 313</a></p> <p>Modified title of <a href="#">Table 55: UART receiver tolerance when UART_DIV[3:0] is zero on page 337</a> and <a href="#">Table 56: UART receiver's tolerance when UART_DIV[3:0] is different from zero on page 338</a></p> <p>Modified RWU bit description in <a href="#">Section 22.7.6: Control register 2 (UART_CR2) on page 367</a></p> <p>Modified <a href="#">Section 23.4.2: Normal mode on page 382</a></p> <p>Added note to FE bit description in <a href="#">Section 22.7.1: Status register (UART_SR) on page 363</a></p> <p>Modified <a href="#">Section 24.9: Reading the conversion result on page 440</a> and <a href="#">Section 24.11.2: ADC data buffer register x low (ADC_DBxRL) (x=or 0..7 or 0..9) (DBL[7:0] instead of DB[7:0])</a></p>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
07-May-2013	9	<p>Added low density STM8AF devices.</p> <p>Replaced all references of STM8A with STM8AF.</p> <p>Updated <a href="#">Table 14: Devices with 4 trimming bits</a>.</p> <p>Updated <a href="#">Section 11.5: Unused I/O pins</a>.</p> <p>Updated Halt/ Active-halt description in <a href="#">Table 68: beCAN behavior in low power modes</a>.</p> <p>Added UART4 in <a href="#">Section 22: Universal asynchronous receiver transmitter (UART)</a>.</p> <p>Updated <a href="#">Section 23.4.4: Time triggered communication mode</a>.</p>
19-Jun-2014	10	<ul style="list-style-type: none"> <li>- <a href="#">Introduction</a>: updated the description of the medium and high density STM8AF devices.</li> <li>- <a href="#">Section 4.4.1: STM8S and STM8AF memory organization</a>: updated the description of Medium density STM8AF devices.</li> <li>- <a href="#">Section 23: Controller area network (beCAN)</a>: changed the default reset value for CAN_MSR.</li> <li>- <a href="#">Section 6: Interrupt controller (ITC)</a>: replaced IRS_CC with ISR_CC in the footnote of <a href="#">Table 9: Interrupt enabling/disabling inside an ISR</a>.</li> <li>- <a href="#">Section 11: General purpose I/O ports (GPIO)</a>: removed the footnote (1) in <a href="#">Table 21: I/O port configuration summary</a></li> <li>- <a href="#">Section 24: Analog/digital converter (ADC)</a>: replaced the first sentence in <a href="#">Section 24.5.6: Analog watchdog</a>, and replaced AWDENx with AWENx in the heading of <a href="#">Table 74</a>.</li> <li>- <a href="#">Section 21: Inter-integrated circuit (I2C) interface</a>: added the section “SCL master clock generation”, and updated <a href="#">Section 21.4.2: I2C master mode</a>, <a href="#">Section 21.7.3: Frequency register (I2C_FREQR)</a> and <a href="#">Section 21.7.13: TRISE register (I2C_TRISER)</a>.</li> <li>- <a href="#">Section 24.5.3: Channel selection</a>: corrected a typo on ACD_CSR.</li> <li>- Updated <a href="#">Section 11.7.4: Schmitt trigger</a>.</li> </ul>
23-Mar-2015	11	<p>Updated:</p> <ul style="list-style-type: none"> <li>- the document title and the <a href="#">Introduction</a> on the cover page,</li> <li>- the very first sentence in <a href="#">Section 17.4.6: Synchronization between TIM1, TIM5 and TIM6 timers</a>,</li> <li>- TS[2:0] bit definitions in <a href="#">Section 17.7.3: Slave mode control register (TIM1_SMCR)</a>,</li> <li>- TS[2:0] bit definitions in <a href="#">Section 18.6.3: Slave mode control register (TIM5_SMCR)</a></li> <li>- TS[2:0] bit definitions in <a href="#">Section 19.6.3: Slave mode control register (TIM6_SMCR)</a></li> <li>- <a href="#">Section 17.4.6: Synchronization between TIM1, TIM5 and TIM6 timers</a>: replaced “TS = 001 in the TIMx_SMCR register” with “see TS[2:0] bit definitions in TIMx_SMCR register.”</li> <li>- <a href="#">Section 23.4: Operating modes</a>: added a note about PG0 port,</li> <li>- the Disclaimer.</li> </ul>

**Table 79. Document revision history (continued)**

Date	Revision	Changes
23-Sep-2015	12	<p><b>Window watchdog (WWDG)</b></p> <ul style="list-style-type: none"> <li>– Corrected the downcounter bit number in <a href="#">Figure 28: Watchdog block diagram</a></li> </ul> <p><b>Inter-integrated circuit (I2C) interface</b></p> <ul style="list-style-type: none"> <li>– Removed PEC calculation and PEC register boxes in <a href="#">Figure 101: I2C block diagram</a>.</li> </ul> <p><b>Universal asynchronous receiver transmitter (UART)</b></p> <ul style="list-style-type: none"> <li>– Updated the LIN version in <a href="#">Section 22.1: Introduction</a>:</li> </ul> <p><b>Controller area network (beCAN)</b></p> <ul style="list-style-type: none"> <li>– Improved the readability of <a href="#">Figure 148</a> to <a href="#">Figure 151</a>.</li> </ul>
25-Jul-2017	13	<ul style="list-style-type: none"> <li>– Updated the introduction of the <a href="#">Section 22.4.3: Slave mode with automatic resynchronization enabled</a>.</li> <li>– Updated <a href="#">Section 5.3: SWIM modes</a>.</li> <li>– All figures of the document were updated (format update, no content update). Below figures were redrawn: <a href="#">Figure 13</a>, <a href="#">Figure 14</a>, <a href="#">Figure 25</a>, <a href="#">Figure 26</a>, <a href="#">Figure 27</a>, <a href="#">Figure 28</a>, <a href="#">Figure 29</a>, <a href="#">Figure 34</a> and <a href="#">Figure 37</a>.</li> <li>– For the whole document, chapter titles were updated to be more specific about their content whenever it was relevant.</li> </ul>
04-Oct-2017	14	<p>Updated:</p> <ul style="list-style-type: none"> <li>– <a href="#">Section 6.2.2: Interrupt sources</a></li> <li>– <a href="#">Figure 13: Interrupt processing flowchart</a></li> <li>– <a href="#">Figure 15: Concurrent interrupt management</a></li> <li>– <a href="#">Figure 16: Nested interrupt management</a></li> <li>– <a href="#">Figure 100: I2Cbus protocol</a></li> <li>– <a href="#">Figure 105: Method 1: transfer sequence diagram for master receiver</a></li> <li>– <a href="#">Figure 106: Method 2: transfer sequence diagram for master receiver when N &gt;2</a></li> <li>– <a href="#">Figure 107: Method 2: transfer sequence diagram for master receiver when N=2</a></li> <li>– <a href="#">Figure 108: Method 2: transfer sequence diagram for master receiver when N=1</a></li> </ul>

# Index

## A

ADC_AWCRH .....	452
ADC_AWCRL .....	452
ADC_AWSRH .....	451
ADC_AWSRL .....	451
ADC_CR1 .....	444
ADC_CR2 .....	445
ADC_CR3 .....	446
ADC_CSR .....	443
ADC_DBxRH .....	441
ADC_DBxRL .....	442
ADC_DRH .....	447
ADC_DRL .....	447
ADC_HTRH .....	449
ADC_HTRL .....	449
ADC_LTRH .....	450
ADC_LTDL .....	450
ADC_TDRH .....	448
ADC_TDRL .....	448
AWU_AP1 .....	119
AWU_CSR1 .....	118
AWU_TBR .....	119

## B

BEEP_CSR .....	122
----------------	-----

## C

CAN_BTR1 .....	411
CAN_BTR2 .....	412
CAN_DGR .....	408
CAN_ESR .....	409
CAN_FCR1 .....	420
CAN_FCR2 .....	421
CAN_FCR3 .....	422
CAN_FiRx .....	423
CAN_FMR1 .....	418
CAN_FMR2 .....	419
CAN_IER .....	407, 410
CAN_MCR .....	401
CAN_MCSR .....	413
CAN_MDAR .....	417
CAN_MDLCR .....	416
CAN_MFMIR .....	414
CAN_MIDR1 .....	415
CAN_MIDR2 .....	415
CAN_MIDR3 .....	416

CAN_MIDR4 .....	416
CAN_MSR .....	402
CAN_MTSRH .....	417
CAN_MTSRL .....	417
CAN_PSR .....	408
CAN_REC1 .....	411
CAN_RFR .....	406
CAN_TECR .....	410
CAN_TPR .....	404
CAN_TSR .....	403
CFG_GCR .....	28
CLK_CCOR .....	98
CLK_CKDIVR .....	94
CLK_CMSR .....	92
CLK_CSSR .....	97
CLK_ECKR .....	91
CLK_HSITRIMR .....	99
CLK_ICKR .....	90
CLK_PCKENR1 .....	95
CLK_PCKENR2 .....	96
CLK_SWCR .....	93
CLK_SWR .....	92

## E

EXTI_CR1 .....	70
EXTI_CR2 .....	71

## F

FLASH_CR1 .....	51-52, 55
FLASH_NCR2 .....	53

## I

I2C_CCRH .....	316
I2C_CCRL .....	315
I2C_CR1 .....	305
I2C_CR2 .....	306
I2C_DR .....	309
I2C_FREQR .....	307
I2C_ITR .....	314
I2C_OARH .....	308
I2C_OARL .....	308
I2C_SR1 .....	310
I2C_SR2 .....	312
I2C_SR3 .....	313
I2C_TRISER .....	318
ITC_SPRX .....	69

IWDG_KR .....	126	TIM1_IER .....	194
IWDG_PR .....	126	TIM1_OISR .....	218
IWDG_RLR .....	127	TIM1_PSCRH .....	209
		TIM1_PSCRL .....	209
<b>P</b>		TIM1_RCR .....	210
Px_CR1 .....	112	TIM1_SMCR .....	191
Px_CR2 .....	113	TIM1_SR1 .....	195
Px_DDR .....	112	TIM1_SR2 .....	196
Px_IDR .....	111	TIM4_ARR .....	260
Px_ODR .....	111	TIM4_CNTR .....	259
		TIM4_CR1 .....	255
<b>R</b>		TIM4_CR2 .....	256
RST_SR .....	77	TIM4_EGR .....	259
<b>S</b>		TIM4_IER .....	258
SPI_CR1 .....	282	TIM4_PSCR .....	259
SPI_CR2 .....	283	TIM4_SMCR .....	256
SPI_CRCPR .....	286	TIM4_SR1 .....	258
SPI_DR .....	286	TIMx_ARRH .....	244
SPI_ICR .....	284	TIMx_ARRL .....	245
SPI_RXCRCR .....	286	TIMx_CCER1 .....	241
SPI_SR .....	285	TIMx_CCER2 .....	242
SPI_TXCRCR .....	287	TIMx_CCMR1 .....	236
		TIMx_CCMR2 .....	239
<b>T</b>		TIMx_CCMR3 .....	240
TIM1_ARRH .....	210	TIMx_CCR1H .....	245
TIM1_ARRL .....	210	TIMx_CCR1L .....	246
TIM1_BKR .....	215	TIMx_CCR2H .....	246
TIM1_CCER1 .....	205	TIMx_CCR2L .....	246
TIM1_CCER2 .....	208	TIMx_CCR3H .....	247
TIM1_CCMR1 .....	198	TIMx_CCR3L .....	247
TIM1_CCMR2 .....	202	TIMx_CNTRH .....	242
TIM1_CCMR3 .....	203	TIMx_CNTRL .....	243
TIM1_CCMR4 .....	204	TIMx_CR1 .....	229
TIM1_CCR1H .....	211	TIMx_CR2 .....	230
TIM1_CCR1L .....	211	TIMx_EGR .....	235
TIM1_CCR2H .....	212	TIMx_IER .....	232
TIM1_CCR2L .....	212	TIMx_PSCR .....	244
TIM1_CCR3H .....	213	TIMx_SMCR .....	231
TIM1_CCR3L .....	213	TIMx_SR1 .....	233
TIM1_CCR4H .....	214	TIMx_SR2 .....	234
TIM1_CCR4L .....	214		
TIM1_CNTRH .....	208		
TIM1_CNTRL .....	209		
TIM1_CR1 .....	188		
TIM1_CR2 .....	190		
TIM1_DTR .....	217		
TIM1_EGR .....	197		
TIM1_ETR .....	192		

---

UART\_GTR ..... 373  
UART\_SR ..... 363

**W**

WWDG\_CR ..... 132  
WWDG\_WR ..... 133

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved